Diplomarbeit

im Studiengang Medieninformatik

an der Hochschule der Medien

Virtueller Museumsführer
am Beispiel der Stiftskirche Stuttgart
Eine VR-basierte Edutainment-Anwendung
in der CAVE™

Vorgelegt von

André Oliver Haas

9. März 2005

Erstprüfer: Prof. Dr. Jens-Uwe Hahn

Zweitprüfer: Dipl.-Ing. Matthias Bues

Fraunhofer IAO Institut Arbeitswirtschaft und Organisation

Fachhochschule Stuttgart
Hochschule der Medien

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Stuttgart, Mittwoch, 9. März 2005

André Oliver Haas

Kurzfassung

Diese Diplomarbeit beschreibt die Entwicklung einer Applikation, welche die Stiftskirche Stuttgart für einen virtuellen Museumsbesuch in der CAVE™ aufbereitet. Dabei wird gezeigt, wie das Modell der Stiftskirche für den Einsatz in einer Echtzeitsimulation modelliert wurde und welche Ansprüche dieses Modell und die Texturen erfüllen müssen, um einen realistisch erscheinenden virtuellen Rundgang durch das architektonische Gebäude zu verwirklichen. Um eine intuitive Erkundung der Stiftskirche zu gewährleisten, wurden zwei Navigationsmethoden integriert. Des Weiteren bietet eine geführte Tour die Möglichkeit, das Gebäude gezielt zu erkunden.

Die Stiftskirche ist ein wichtiger Bestandteil der Geschichte Stuttgarts. Daher wurde eine Möglichkeit entwickelt, dem Anwender Informationen zu ausgewählten Objekten in der Stiftskirche (z.B. einige Grabmäler oder die neue Orgel) zu visualisieren. Dadurch kann er sich während des virtuellen Besuchs mit der Geschichte der Stiftskirche auseinandersetzen.

Um zusätzlich ein ganzheitliches Erleben zu erreichen, wurden einige Modelle der Stiftskirche als interaktive Objekte ausgewählt, mit denen der Anwender direkt interagieren kann. Die dafür notwendigen Techniken, die diese Interaktion im virtuellen Raum ermöglichen, wurden in dieser Arbeit entwickelt.

Abstract

This diploma thesis describes the development of an application that processes the "Stiftskirche" (collegiate church) of Stuttgart to enable a virtual museum experience. It demonstrates how the model of the "Stiftskirche" was created for use in a real-time simulation. The requirements this model and the textures have to fulfil in order to present a high visual quality are described. Two methods of navigation have been integrated into the application to deliver an intuitive experience. In addition there is a guided tour available that offers a directed exploration of the building.

The "Stiftskirche" is an important part of the history of Stuttgart. For this reason the capability was developed to present additional information about historically significant objects found in the church (for example, tombs or the new organ). This allows the user to discover some of the historical nature of the building during the virtual visit.

In order to provide an even more holistic experience, some parts of the "Stiftskirche" were chosen to be interactive objects with which the user can interact directly. The specific technologies that enable this interaction in the virtual space are also explained as a part in this thesis.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Übersetzt man den Begriff „Virtual Reality“ (VR) ins Deutsche, so bedeutet er „scheinbare Wirklichkeit“. Dies kling zwar zunächst widersprüchlich, sagt aber genau das aus, was bei diesen Anwendungen geschieht. Die hierbei erzeugte Umgebung ist zwar eine „künstliche Welt“. Trotzdem fühlt der Anwender sich in die realistisch erscheinende Umgebung hineinversetzt, da VR-Systeme eine äußerst detailgetreue und maßstabsgerechte Visualisierung ermöglichen und dem Anwender zusätzlich vielfältige Möglichkeiten der Navigation und Interaktion bieten. Daher eignen sich Systeme der Virtual Reality auch hervorragend zur Visualisierung architektonischer Modelle.

Im Rahmen dieser Diplomarbeit wurde am Beispiel der im Jahr 2004 restaurierten Stuttgarter Stiftskirche eine VR-Anwendung entwickelt, die neben der Visualisierung des Gebäudes auch historische Aspekte vermittelt. Dabei wurde besonderes Augenmerk auf die Benutzerführung zur Navigation, Interaktion und Informationsvermittlung sowie auf die Gestaltung des visuellen Modells der Stiftskirche Stuttgart gelegt.

1.2 Problemstellung

Die Problemstellung lässt sich in mehrere Teile untergliedern.

Der konzeptionelle Teil der Arbeit beinhaltet die Definition der visuellen Modelle und die Festlegung der zu vermittelnden Informationen. Daraus wurden Konzepte zur Navigation im Raum, zur Interaktion mit einzelnen Modellen sowie zur Vermittlung von Informationen entwickelt und realisiert. Erweitert wurde dies um ein Konzept, welches eine geführte Tour durch die Architektur ermöglicht und dabei die informativen Bereiche und Objekte der Stiftskirche hervorhebt.

Bei der Erstellung des Modells (der Stiftskirche) steht die maßstabsgerechte Modellierung der für den Einsatz in Echtzeitumgebungen benötigten Geometrien sowie deren Texturierung im Vordergrund.

Die im Rahmen dieser Arbeit entwickelte Applikation verwirklicht die Konzepte zur Navigation, Interaktion und Informationsvermittlung für die Visualisierung in einer CAVE™.

Ziel dieser Arbeit ist ein virtueller Rundgang durch die Stiftskirche Stuttgart, in welchem der Betrachter sich mit deren Geschichte auseinandersetzen kann und die Möglichkeit zur Interaktion mit einigen ausgewählten Exponaten der Kirche hat.

1.3 Aufbau der Diplomarbeit

In Kapitel 2 wird auf die Grundlagen von Virtual Reality eingegangen. Dies betrifft neben der Beschreibung menschlicher visueller Wahrnehmung und stereoskopischer Darstellung auch unterschiedliche Geräte und Techniken, die in VR-Systemen eingesetzt werden.

Kapitel 3 zeigt die in dieser Arbeit verwendeten Technologien auf. Zum einen wird die Fraunhofer Software Lightning mit ihren Komponenten dabei ausführlich beschrieben. Zum anderen sind die Anforderungen an die Modellierung und Texturierung unter gestalterischen Gesichtspunkten angeführt. Abgeschlossen wird das Kapitel mit der Vorstellung des Datenformats VRML und der verwendeten Datenbank SQLite.

In Kapitel 4 wird anhand einiger Beispiele gezeigt, wie die in der Arbeit verwendeten 3D-Modelle für den Einsatz in virtuellen Umgebungen erstellt wurden. Dabei wird erläutert, welche Ansprüche diese Modelle erfüllen müssen, um eine Echtzeitvisualisierung zu gewährleisten. Des Weiteren wird die Texturierung von Geometrie ausführlich beschrieben.

Kapitel 5 zeigt das entstandene Modell der Stiftskirche Stuttgart und beschreibt die einzelnen geometrischen Modelle, welche in der virtuellen Stiftskirche vorhanden sind. Die Modelle sind hier in unterschiedliche Gruppen aufgeteilt, wobei auf die verschiedenen interaktiven und informativen Modelle ausführlich eingegangen wird.

In Kapitel 6 werden die einzelnen Funktionen der Applikation auf konzeptioneller Ebene beschrieben. Dabei sind die in dieser Arbeit ausgearbeiteten Konzepte zur Navigation im Raum, zur Informationsvermittlung sowie zur Interaktion mit den einzelnen Objekten detailliert geschildert.

Kapitel 7 veranschaulicht die einzelnen Klassen der Softwareimplementierung, mit welchen die Funktionalität für Navigation, Interaktion und Informationsvermittlung programmiert wurde.

In Kapitel 8 werden anhand von ausgewählten, in der CAVE™ gemachten Aufnahmen die Ergebnisse dieser Arbeit vorgestellt.

Kapitel 9 schließt die Arbeit mit der Darstellung einiger Erweiterungsansätze, auf welche nachfolgende Arbeiten aufbauen können, ab.

2 Technische Grundlagen

Um dem Betrachter eine Immersion[1] in die virtuelle Welt zu ermöglichen und dabei z.B. in virtuellen Gebäuden umhergehen oder mit virtuellen Objekten interagieren zu können, ist einiges an Technologie notwendig. Hier gibt es eine Vielzahl von Entwicklungen, wobei die folgenden Abschnitte nur die in dieser Arbeit eingesetzten Technologien beschreiben.

2.1 Stereoskopische Darstellung

Das Sehen mit nur einem Auge wird als monokulares Sehen bezeichnet. Aus einem monokular wahrgenommenen, unbewegten und „flachen" Bild lassen sich Tiefeninformationen entnehmen. Bei der Wahrnehmung der Tiefeninformation ist die gegenseitige Verdeckung von Objekten ein elementarer Bestandteil, da ein verdecktes Objekt dem Betrachter weiter entfernt erscheint als das im Vordergrund liegende. Das Wechselspiel zwischen Licht und Schatten sowie unterschiedliche Größenverhältnisse von Objekten sind an dieser Stelle als weitere monokulare Tiefenkriterien zu erwähnen.

Bewegt sich der Beobachter, kommt die Bewegungsparallaxe als verstärkender Effekt der Tiefenwahrnehmung hinzu. Dies bezeichnet ein Phänomen, durch welches das Gehirn eine Tiefeninformation gewinnen kann. Mit der Bewegung des Betrachters scheinen sich die nahen Objekte schneller zu bewegen als die weiter entfernt liegenden, was bei dem Betrachter die Illusion dreidimensionaler Tiefe hervorruft.

Denselben Effekt erhält man auch beim Sehen mit zwei Augen, welches als binokulares Sehen bezeichnet wird. Zusätzlich entsteht durch die Konvergenz, die Akkommodation und die Querdisparation der Augen ein Tiefeneindruck (stereoskopisches Sehen) [hic95]. Die Konvergenz bezeichnet das Zusammenlaufen der Sehachsen in einem Punkt, auf den die Augen fixiert sind. Bei der Betrachtung eines Objekts im Abstand von ca. 8-9 Meter verlaufen die Blicklinien beider Augen beinahe parallel. Je näher der Gegenstand ist, desto größer ist der Winkel, unter dem sich die Blicklinien schneiden (Konvergenzwinkel). Der Prozess der Fokussierung (Scharfstellen) der Augen auf einen Punkt wird Akkommodation genannt. Da die Augen in einem bestimmten Abstand zueinander liegen (Augabstand), erzeugt ein betrachteter Gegenstand ein leicht unterschiedliches Bild auf der jeweiligen

[1] Der Begriff Immersion (lat. immersio "Eintauchen") bezeichnet in der Virtual Reality, psychisches und physisches Eintauchen in die virtuelle Umgebung, bzw. die Auflösung von räumlichen Grenzen.

Netzhaut der beiden Augen. Dieser Effekt wird als Querdisparation bezeichnet. Das Gehirn ist in der Lage, daraus Tiefeninformationen zu gewinnen.

Die dreidimensionale Darstellung wird häufig über gemeinsame Projektionen der Bilder für das rechte und linke Auge auf eine Fläche realisiert. Dadurch wird eine korrekte Konvergenz der Augen auf einen bestimmten Punkt an einem Objekt im Raum erreicht. Die Akkommodation wird hierbei jedoch nicht berücksichtigt, da die Augen nicht das Objekt, sondern die Projektionsfläche fokussieren (siehe Abbildung 1). Trotzdem wird mit dieser Darstellungsart ein guter 3D-Eindruck hervorgerufen.

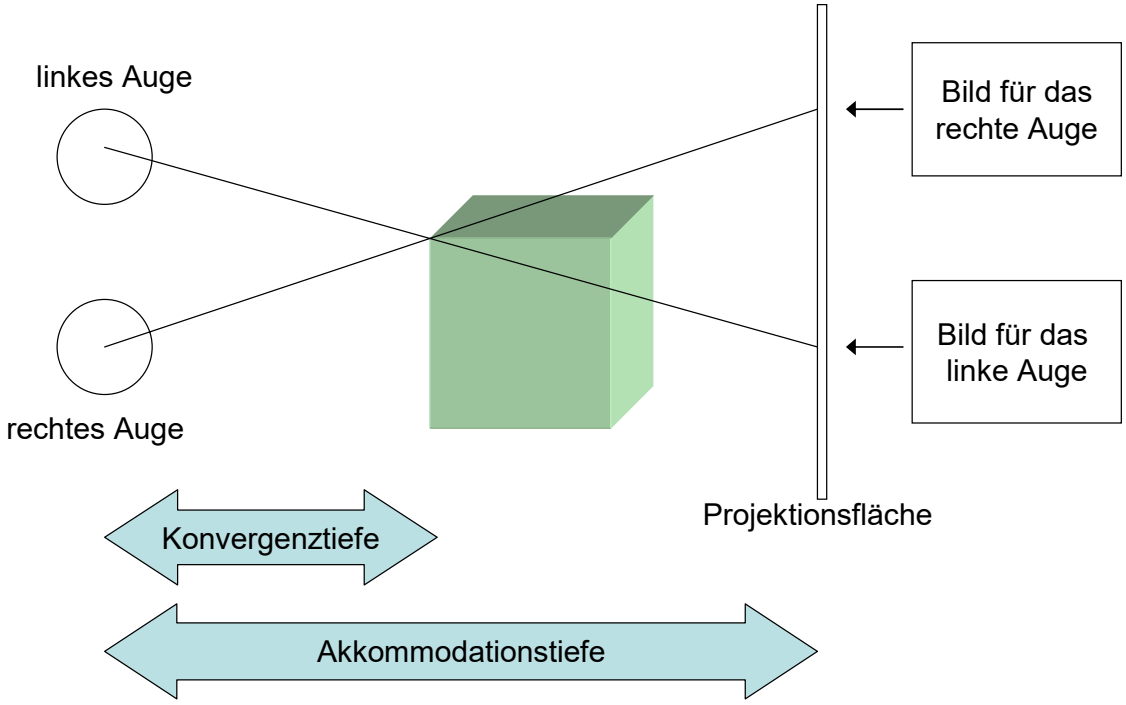


Abbildung 1: Stereoskopische Projektion auf eine Fläche.

Zur Trennung der beiden Bilder (Kanäle) für das rechte und linke Auge gibt es mehrere unterschiedliche Verfahren. Folgend werden zwei Verfahren zur Kanaltrennung über Polarisationsfilter (passiv Stereo) oder Shutterbrillen (aktiv Stereo) vorgestellt.

Passiv Stereo benötigt zur Darstellung zwei Projektoren. Das von den Projektoren ausgestrahlte Licht wird mittels Polfilter polarisiert. Die Kanaltrennung erfolgt in den Brillen ebenfalls über Polfilter, damit jedes Auge nur das für es bestimmte Bild zu sehen bekommt. Hierbei gibt es zwei unterschiedliche Möglichkeiten das Licht zu polarisieren. Wird das Licht des Projektors für das linke Auge senkrecht und für das rechte waagerecht linear polarisiert, kann es durch entsprechende Polfilter in den Brillen wieder getrennt werden. Die Bildtrennung wird hierbei jedoch vom Neigungswinkel des Kopfes beeinflusst, was sich für eine Vielzahl von Anwendungen, bei denen sich der Betrachter frei bewegen kann,

als nachteilig erweist. Zirkulare Polarisation kennt dieses Problem jedoch nicht, denn das Licht wird hierbei entweder links- oder rechtsdrehend polarisiert, wodurch die Kopfneigung keinerlei Einfluss auf die Darstellung hat.

Bei der **Aktiven Stereoskopischen Darstellung** wird zur Projektion der Bilder für das linke und rechte Auge nur ein Projektor benötigt. Dieser erhält vom Rendersystem abwechselnd ein Bild für jedes Auge, welches mit einer Frequenz von 100 Hz bis 120 Hz dargestellt wird. Dabei verdeckt eine Shutterbrille im Wechsel mit derselben Frequenz jeweils das eine oder andere Auge. Durch die hohe Wiederholrate des Öffnens und Schließens der Brille entsteht aufgrund der Trägheit des Auges ein flimmerfreies dreidimensionales Bild. Nachteilig hierbei sind das hohe Gewicht und die Anschaffungskosten der Brillen. Vorteilhaft ist allerdings, dass nur ein Projektor pro Projektionswand benötigt wird.

Abbildung 2: Aktive und passive Brillen.

2.2 Ausgabegeräte

Alle Geräte, die auf einen der menschlichen Sinne Einfluss nehmen können, zählen zu den Ausgabegeräten. Da das Sehen der wichtigste Sinn des Menschen ist und somit eine entsprechend große Rolle bei der Wahrnehmung der Umwelt spielt, stellen visuelle Ausgabegeräte den Hauptbestandteil eines VR-Systems dar. Diese sind zuständig für die Vermittlung von bildhaften Inhalten an den Betrachter. Um den Immersionsgrad des Anwenders zu steigern können bis zu einem gewissen Grad ebenfalls akustisches und haptisches (fühlendes) Feedback in ein VR-System integriert werden. Herkömmliche Ausgabegeräte wie z.B. ein Computermonitor können dem Benutzer nicht das Gefühl vermitteln, sich innerhalb einer virtuellen Welt zu befinden. Daher wurden spezielle Ausgabegeräte entwickelt, von denen im Folgenden jedoch nur die am weitesten verbreiteten Ausgabegeräte erläutert werden. Eine große Anzahl der unterschiedlichsten Ausführungen solcher Geräte sowie ausführliche Beschreibungen dieser sind unter [she03] und [bow04] zu finden.

Ein **Head Mounted Display** (HMD) besteht aus zwei kleinen Bildschirmen (Displays), die jeweils vor einem Auge positioniert sind (s. Abbildung 3). Um die Augen des Betrachters von Fremdlicht abzuschirmen, sind sie in einem helmähnlichen Gerät untergebracht. Als Bildschirme werden meist Flüssigkristallbildschirme verwendet, da diese leichter und platzsparender als herkömmliche Kathodenstrahlröhren sind. Die Displays befinden sich nur vor, nicht aber seitlich der Augen, wodurch das Blickfeld des Benutzers nicht vollständig ausgefüllt wird. Die Größe des Blickfeldes ist von der Größe der eingesetzten Displays sowie von der verwendeten Optik abhängig. Je kleiner dabei das Blickfeld ist, desto mehr hat der Betrachter den Eindruck, durch eine Röhre zu blicken. Mittels Trackern, welche am HMD befestigt sind, wird die Position und Orientierung des Kopfes registriert, sodass das dargestellte Bild an die Position und Orientierung des Benutzers angepasst werden kann.

Abbildung 3: Head Mounted Display.

Eine **CAVE™** (Cave Automated Virtual Environment) [cru93] ist ein Raum zur Projektion einer Illusionswelt der virtuellen Realität. Die Bezeichnung „Cave“ rekurriert bewusst auf das Höhlengleichnis in Platons „Republik“, welches sich mit dem Verhältnis von Wahrnehmung und Erkenntnis sowie Realität und Illusion beschäftigt [spe96].

Die CAVE™ erlaubt die interaktive räumliche Visualisierung von hochkomplexen Grafikdaten in einem kubischen Raum, der eine Kantenläge von ca. 2.5 bis 3 Meter besitzt. Dabei gibt es verschieden komplexe Ausführungen von CAVE™ Systemen. Diese reichen von einer Minimalkonfiguration von 3 bis zu einer vollen Ausbaustufe von 6 Wänden, auf welche projiziert werden kann. Bei den häufigsten Konfigurationen bestehen die Wände

des Raums aus diffusen, transparenten Kunststoffscheiben, welche mittels Rückprojektion je nach Stereoverfahren (aktiv oder passiv) von bis zu 2 Projektoren bestrahlt werden.

In der vollen Ausbaustufe von 6 Seiten ist der Anwender komplett von Projektionsflächen umgeben. Seine Position und Orientierung sowie die der Eingabegeräte werden durch ein magnetisches Trackingsystem erfasst, da es in diesem geschlossenen Raum nur sehr schwer möglich ist, ein optisches Trackingsystem zu installieren. Dadurch kann der Anwender in der virtuellen Umgebung (Virtual Environment VE) frei in allen Achsen und Richtungen agieren, wodurch der maximale Grad der Immersion für den Betrachter erreicht wird.

Das CCVE (Competence Center Virtual Environments) des Fraunhofer IAO eröffnete am 16. Mai 2001 die erste 6-Seiten-CAVE „HyPI-6"[2]. In Bezug auf Projektionstechnik und Computerhardware wird eine neue Qualität, Flexibilität und Skalierbarkeit von Projektionssystemen für virtuelle Umgebungen erreicht. Aktive und passive Stereoprojektion sowie eine Ansteuerung mit einem PC Clustersystem oder einem High End Grafikcomputer (Onyx 3 IR) erweitern das Konzept zu einer High End Lösung. Durch den parallelen Einsatz der hier beschriebenen unterschiedlichen Technologien wird dabei von einem hybriden System gesprochen.

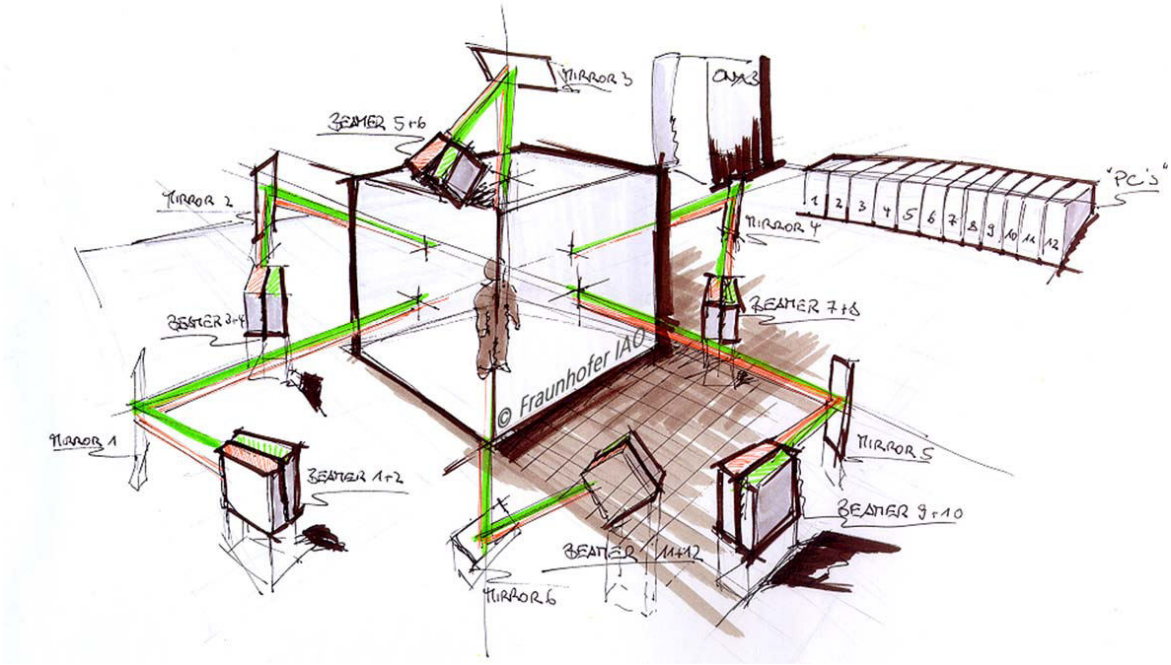


Abbildung 4: Schematischer Aufbau der „HyPI-6“ am Fraunhofer IAO Stuttgart.

[2] Hybrides Personal Immersion System, 6-seitig

2.3 Trackingsysteme

> *„In vielen 3D-Anwendungen ist es wichtig, dass durch das Userinterface notwendige Informationen bereitgehalten werden, wie den Standort des Anwenders oder den von physischen Objekten im Raum. Beispielsweise kann eine Applikation die Position und Orientierung des Anwenders benötigen, damit die volle Bewegungsparallaxe und räumliche Tiefe in der Applikation mit einbezogen werden kann." [bow04]*

Damit dem Betrachter immer ein geometrisch korrektes Bild berechnet und dargestellt werden kann, ist es unerlässlich, laufend die aktuelle Position und Orientierung des Betrachters an das System zu übermitteln, denn die korrekte Darstellung der Szene ist abhängig von dem aktuellen Standpunkt und der Blickrichtung des Anwenders. Um dies zu gewährleisten ist ein Trackingsystem notwendig, welches sozusagen das Bindeglied zwischen physischer und virtueller Realität bereithält.

Ein Trackingsystem erhöht den Immersionsgrad wesentlich und verbessert die Interaktionsfähigkeit des Betrachters mit der virtuellen Umgebung. Es dient der Bestimmung von aktueller Position und Blickrichtung des Betrachters und der von ihm verwendeten Interaktionsgeräte im Raum und ist deshalb auch als Motion Tracking bekannt. Bowman führt aus, dass ein präzises Tracking mit kurzer Latenzzeit[3] ein entscheidender Faktor ist, um verwendbare Interaktionen mit virtuellen Umgebungen zu erreichen (vgl. [bow04]). Derzeit gibt es mehrere Arten von Trackingsystemen wie z.B. mechanisches, akustisches, magnetisches und optisches Tracking. Anwendung finden jedoch hauptsächlich die beiden letzteren Verfahren.

Optische Systeme arbeiten mit Lichtsendern, welche infrarotes Licht ausstrahlen. Hierbei wird Licht an Reflektoren, die an der Shutterbrille oder am Interaktionsgerät des Anwenders in einem charakteristischen Muster befestigt sind, zurückgestrahlt. Die Reflektoren werden je nach System von 2 bis 4 Kameras erfasst. Durch die charakteristische Anordnung können nun die Position und Orientierung des jeweiligen Gerätes errechnet werden. Optische Systeme arbeiten um ein Vielfaches genauer als magnetische Systeme. Zudem benötigen sie keine Kabel, wodurch ein spürbar freieres Arbeiten möglich wird. Jedoch sind optische Systeme derzeit noch um einiges kostspieliger und eingeschränkt in ihrer Installationsmöglichkeit, da sie einen direkten Sichtkontakt zwischen Sender und Empfänger benötigen.

---

[3] Antwortzeit, die vergeht zwischen dem Zeitpunkt der Änderung der Position und dem Zeitpunkt, an dem das System diese Änderung erhält.

Abbildung 5: Optisch getrackte Shutterbrille.

Magnetische Systeme verwenden einen Sender, welcher ein langsam frequentiertes Feld emittiert. Ein kleiner Empfänger bestimmt seine eigene Position und Orientierung relativ zu dem vom Sender ausgesendeten Magnetfeld. Dieses Trackingsystem findet besonders in einer 6 Seiten CAVE™ Verwendung, da dort kein Sichtkontakt zwischen Sender und Empfänger vorhanden ist.

2.4 Interaktions- und Eingabegeräte

Interaktionen mit Objekten in virtuellen Umgebungen benötigen spezielle Eingabegeräte. Diese unterscheiden sich von herkömmlichen PC Eingabegeräten im Wesentlichen dadurch, dass sie im Raum (also mit 6 Freiheitsgraden) bewegt werden können. Aus diesem Bedarf heraus ist in den letzten Jahren eine Fülle von verschiedenartigen Eingabegeräten entstanden, was Kettner dazu veranlasste, von einem ganzen *Zoo von höherdimensionalen Eingabegeräten* zu sprechen [ket93]. Über die gängigsten Technologien von Eingabegeräten soll im Folgenden ein kurzer Überblick gegeben werden. Detaillierte Beschreibungen von Eingabegeräten liefert Bowman [bow04].

Der **Datenhandschuh** (Data Glove) ist das wohl bekannteste VR-Interaktionsgerät. Er dient der Erfassung von Hand- und Fingerbewegungen im Raum. Dadurch entsteht die Möglichkeit, die Bewegungen der „echten“ Hand auf eine virtuelle Hand zu übertragen und somit virtuelle Objekte zu manipulieren oder mittels bestimmter Gesten verschiedene

Aktionen auszulösen. Ein Data Glove besitzt je nach Ausführung bis zu 22 Dehnmessstreifen, die zur Ermittlung der Fingerhaltung genutzt werden. Seine aktuelle Position und Orientierung innerhalb der Szene wird über das bereits erwähnte Trackingsystem bestimmt.

Die **Flying Mouse** ist nach dem Datenhandschuh ein weiteres weit verbreitetes Eingabegerät für VR-Simulationen. Der Anwender hält dieses Eingabegerät in der Hand und kann über mehrere Schalter verschiedene Aktionen auslösen. Die Position und Orientierung des Eingabegerätes im Raum wird in allen 6 Freiheitsgraden durch das Trackingsystem erfasst. Im virtuellen Raum ist dieses Eingabegerät für die Navigation sowie für das Greifen und Selektieren virtueller Objekte geeignet. Aufgrund der an Flügeln angebrachten Reflektoren für das optische Tracking hat die Eigenentwicklung am CCVE den Namen "Hornet" erhalten (s. Abbildung 6). Sie besitzt einen ergonomischen Griff und mehrere Schalter, welche sehr gut mit Daumen und Zeigefinger zu bedienen sind. Die in dieser Arbeit implementierten Walk-, Fly- und unterschiedlichen Interaktionsmetaphern lassen sich damit hervorragend steuern und bedienen.

Im Gegensatz zu den oben genannten Geräten ist die **Space-Mouse** ein tischgebundenes 3D-Eingabegerät. Da der Anwender seinen Arm auf dem Tisch auflegen kann, garantiert sie ein ermüdungsfreieres Arbeiten und ist deshalb als Desktop VR-Eingabegerät gedacht. Durch die spezielle Konstruktion einer beweglichen Sensorkappe, welche sich in allen 6 Freiheitsgraden bewegen lässt, sind damit alle Möglichkeiten von Bewegungen und Rotationen im Raum kontrollierbar. Die Space-Mouse verwendet dabei nicht wie ein getracktes Eingabegerät absolute Koordinaten, sondern steuert einen Geschwindigkeitsvektor in Abhängigkeit der Position und Orientierung der Sensorkappe.

Abbildung 6: Interaktionsgeräte: Hornet und Datenhandschuh.

3 Eingesetzte Technologien

In Kapitel 2 wurde die technische Hardware, welche die Grundlage von Simulationen in Virtual Reality bildet, erläutert. Jedoch sind die bisher dargelegten technischen Anlagen nur der Rahmen für eine Applikation, welche sich dieser Technik bedient. Zusätzlich sind ansprechende 3D-Modelle besonders bei VR-Anwendungen unerlässlich, um den Grad der Immersion des Benutzers zu erhöhen. Die im folgenden Kapitel ausführlich beschriebenen Softwarekomponenten sind somit ein elementarer Bestandteil dieser Diplomarbeit.

3.1 Lightning: Ein System für immersive Echtzeit-Umgebungen

Das Virtual Reality System Lightning ist ein sehr mächtiges Framework für die Entwicklung von Applikationen zur Darstellung von interaktiven und immersiven virtuellen Umgebungen in Echtzeit. Durch die Kombination eines verallgemeinerten Renderkonzeptes und eines Paradigmas zur Verbreitung von Ereignissen lassen sich mit Lightning Applikationen mit komplexer Funktionalität programmieren. Abbildung 7 soll verdeutlichen, wie die Kommunikation zwischen Benutzer, Ein- und Ausgabegeräte und das System selbst zusammen hängen, und so eine Simulationsschleife gewährleisten.

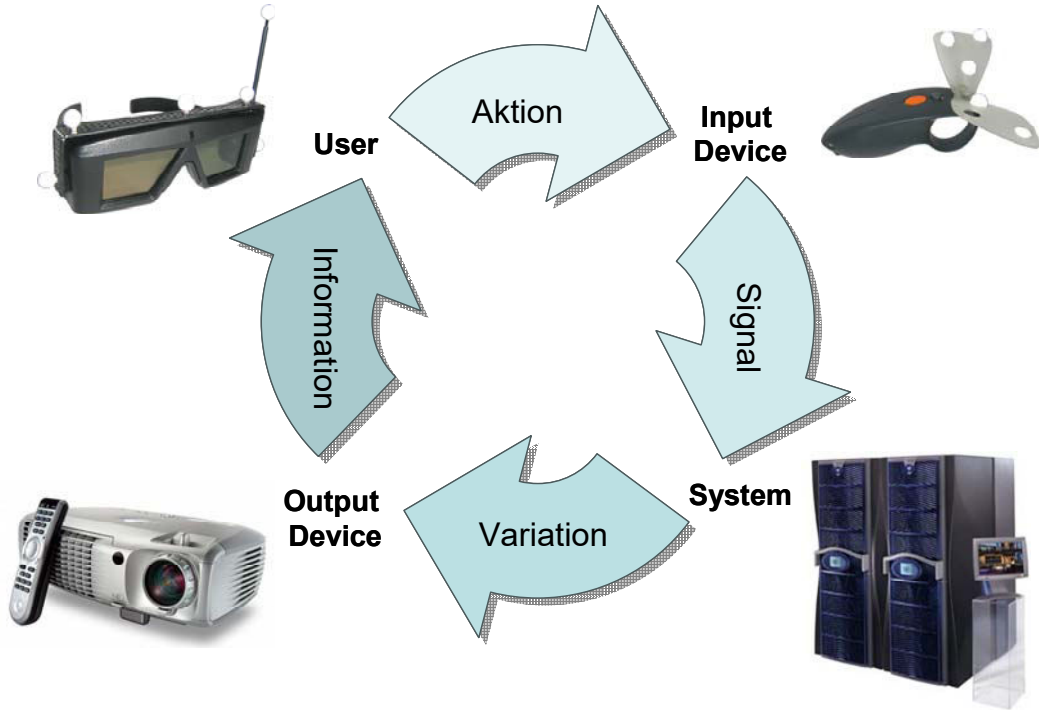


Abbildung 7: Simulationsschleife in VR-Systemen.

Entwickler von Applikationen können von den in den folgenden Unterkapiteln erläuterten Systemkomponenten des Frameworks profitieren, welches eine Eigenentwicklung des Fraunhofer CCVE ist.

3.1.1 Architektur von Lightning

Das Kernstück der Software ist der Objekt Pool, eine Datenbank, in welcher der Entwickler die in einer Szene vorhandenen Objekte (Sensoren, Geometrieobjekte, Lichter, Materialien, Funktionsobjekte, Soundobjekte, Kameraobjekte, Skriptobjekte …) definiert. Um zum Beispiel eine Interaktionsmöglichkeit zu realisieren, muss das Verhalten der jeweiligen Objekte miteinander kombiniert und unterschiedliche Berechnungen durchgeführt werden können.

Dazu besitzen Lightning-Objekte folgende Eigenschaften:

- Vereinheitlichte Schnittstellen, so genannte Felder, die einem Objekt zur Verbreitung von Ereignissen (event propagation) ein standardisiertes Interface zu Verfügung stellen.
- Eine einheitliche Updatefunktion, welche gemäß den Daten am Eingangsfeld den inneren Zustand des Objekts verändert.
- Ausführen der Updatefunktion nur, wenn die Daten am Eingangsfeld verändert wurden. Dies ist eine verallgemeinerte Implementation des so genannten 'Sensors' bei VRML2.0[4] (siehe [car97]).

In Lightning gibt es drei Arten von Objekttypen. Eingebaute Objekte (z.B. der Renderer), dynamisch erzeugte Objekte (z.B. Kameras) sowie Sensoren und Skripte. Die einzelnen Objekte stellen die Verbindung zu darunter liegenden Systemdiensten her. Damit erhält ein Softwareentwickler sehr einfach die Möglichkeit, das System durch eigene Objekte mit beliebiger Funktionalität zu ergänzen. Dazu lassen sich die Objekte mit Skripten, Eingabegeräten wie Motion-Tracker und Spracherkennungssystemen oder Ausgabegeräten wie Displays oder akustischen Systemen mittels Routen (s. Kapitel 3.1.2) verbinden.

---

4 VRML = Virtual Reality Modeling Language

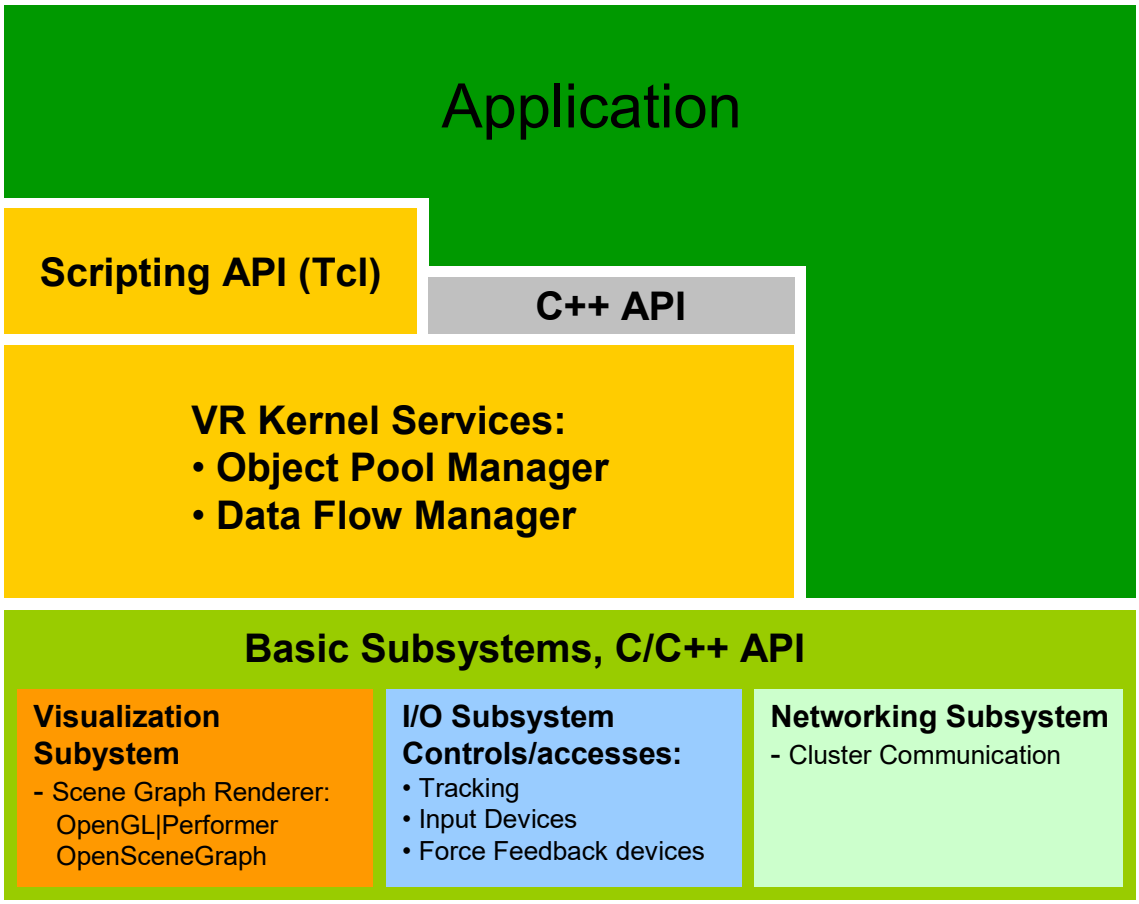


Abbildung 8: Übersicht der Lightning Architektur [bue03].

Mehrere Module zur Kontrolle (Manager) halten Funktionen für eine interne Administration und Kapselung der Objekte bereit. Die Manager sind vor dem Entwickler verborgen, da dieser sie bei der Programmierung nicht in die Applikation einbeziehen muss und somit auch keinen direkten Zugriff darauf hat.

- Der **Objekt Manager** erzeugt (löscht) die Objekte im Pool und verwaltet diesen.
- Der **Routen Manager** propagiert die an den Feldern der Objekte auftretenden Veränderungen mittels Routen durch das System (s. Kapitel 3.1.2).
- Der **Device Manager** erhält die Daten der Trackingsysteme und Eingabegeräte.

3.1.2 Das Routing Konzept

Das Verhalten der Lightning Applikation wird durch das Verhalten der einzelnen Objekte definiert. Um zu erreichen, dass die Objekte Daten von anderen erhalten oder ihre eigenen an andere Objekte weitergeben können, benötigen sie:

- Eingabefelder, die das Verhalten des Objekts steuern,

- das Verhalten, also die Logik des Objekts, welches durch die so genannte „Behaviour-Funktion“ definiert wird,
- Ausgabefelder, die das Ergebnis weitergeben.

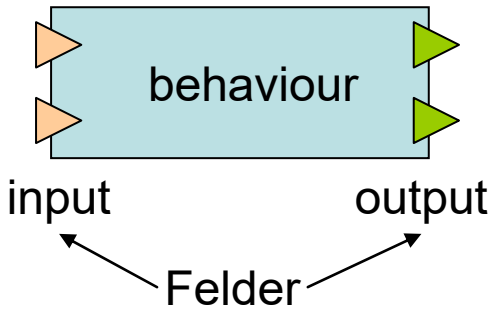


Abbildung 9: Aufbau eines typischen Lightning-Objekts.

Es lassen sich die Ausgabefelder des einen Objekts mit den Eingabefeldern des anderen verbinden, sofern diese den gleichen Typ (z.B. Vec3, Float, String) besitzen. Diese Verbindungen sind gerichtet und werden Routen genannt, wobei ein Eingabefeld (Inputfeld) nur eine, ein Ausgabefeld (Outputfeld) jedoch mehrere Verbindungen eingehen kann. Somit besteht der Datenfluss nur aus Routen und Feldern. Die Ausführung der aktiven Routen wird durch den Routen Manager vollzogen. Dieser sortiert die Routen in einen Baum, in welchem jeder Knoten ein Objekt und jede Kante eine aktive Route darstellt. Um die Objektdaten zu aktualisieren wird pro Frame einmal durch den gesamten Baum traversiert. Änderungen von einem unteren Knoten im Baum auf einen darüber liegenden Knoten haben erst im nächsten Frame eine Auswirkung auf das Objekt. Schleifen im Datenfluss führen daher nicht zu einem Deadlock.

Das folgende vereinfache Beispiel und die dazugehörige Abbildung 10 sollen das dargelegte Prinzip des Routings verdeutlichen.

1. Ein Sensor oder Skript verändert den Wert seines Ausgabefeldes.
2. Die Werte der Eingabefelder anderer Objekte, welche mit dem geänderten Ausgabefeld des Sensors verbunden sind, werden mit einem Flag markiert.
3. Der Routen Manager ruft in Abhängigkeit vom Routen Baum die Behaviour-Funktion aller Objekte auf.
4. Während die Funktionen ausgeführt werden, prüft das Objekt seine geänderten Eingabefelder und berechnet, falls notwendig, die neuen Ausgabewerte.

In diesem Beispiel ist der Sensor der oberste Knoten (Root) des aufgebauten Routen Baumes. Dieser ändert seinen Ausgabewert, welcher durch eine Route mit den Eingabefeldern von Objekt 1 und Objekt 2 verbunden ist. Objekt 2 löst nun Objekt 3 aus. Die Objekte 4 und 5 werden schließlich am Ende des Durchlaufs aufgerufen. Ein Wechsel am Ausgang von Objekt 4 wird jedoch erst im nächsten Frame seine Auswirkung auf das Gesamtsystem zeigen können, um einen Deadlock, ausgelöst durch eine Schleife im System, zu vermeiden.

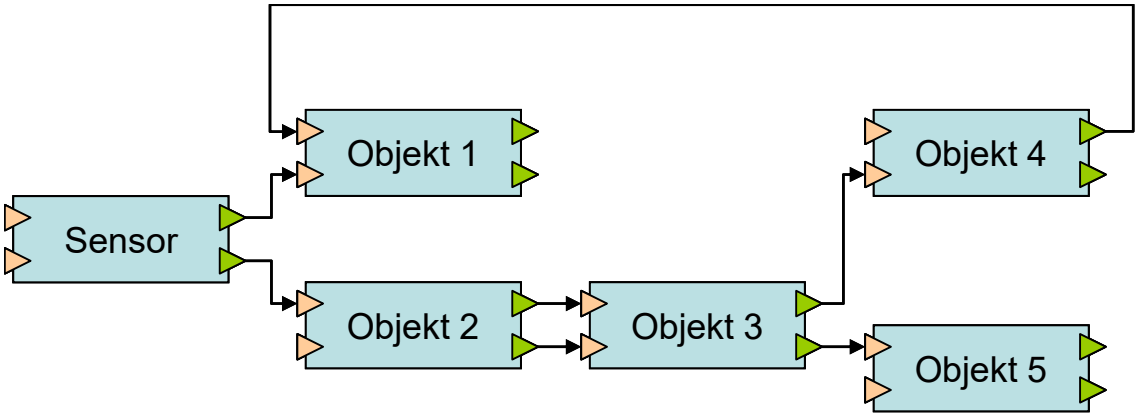


Abbildung 10: Vereinfaches Routing Beispiel.

3.1.3 Anbindung an den Szenengraph

Der Szenengraph OpenGL Performer [roh94] dient der Darstellung der in Lightning geladenen visuellen Geometrie im virtuellen Raum. Alle Objekte einer Szene werden in einem hierarchischen Baum strukturiert abgelegt und deren Attribute in deren jeweiligen Knoten abgespeichert. Damit ist es möglich, Änderungen, die nur einige Knoten und deren Kindknoten im Baum betreffen, zu erkennen und somit nur diese neu zu berechnen. Zur Darstellung der Geometrie greift Performer auf OpenGL eine Ebene darunter zurück. OpenGL ist sozusagen das Interface zwischen der Grafikhardware und dem Szenengraph.

Des Weiteren ist Performer zuständig für die Positionierung der Lichter innerhalb der Szene, die Berechnung von Transformationen und die Darstellung von Materialien und Texturen der vorhandenen Geometrien. Zusätzlich führt er die Berechnung spezieller Operationen wie z.B. das Erkennen von Überschneidung einzelner Grafikobjekte sowie die Darstellung von Partikelsystemen oder Schnittebenen durch. Eine genaue Beschreibung der genannten Funktionalitäten von Performer ist im White Paper von SGI [SGI04] zu finden.

Um Lightning-Objekte mit ihren zugeordneten visuellen Objekten im Szenengraf zu verbinden, besitzt jedes Lightning-Objekt einen Zeiger auf das zugehörige Szenengraph

Objekt. Die Erzeugung des Szenengraph Objekts (z.B. zum Laden externer Geometrie) erfolgt mit dem Lightning-Objekt ltVisObj.

Die Veränderung eines Lightning-Objekts, welches einen Repräsentanten im Szenengraph besitzt (z.B. die Änderung der Position und Orientierung eines geometrischen Objekts oder die Veränderung von Lichtverhältnissen in der Szene), wird durch dessen Behaviour-Funktion an den Szenegraphen weitergegeben.

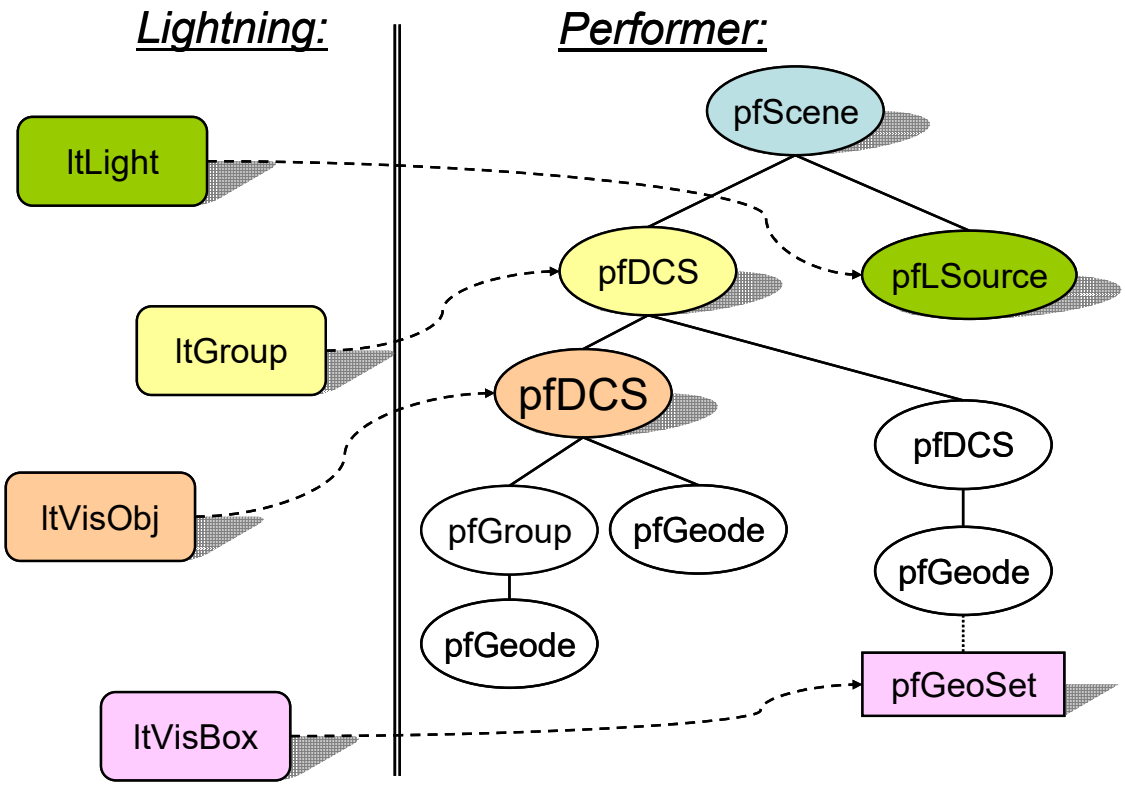


Abbildung 11: Verbindung von Lightning und Szenengraph Objekten [bue03].

3.1.4 Die Programmierschnittstelle [incrTCL]

Nach Blach [bla98] basiert Lightning nicht auf einer bestimmten Sprache, sondern unterstützt eine heterogene Kombination verschiedener Sprachen. Dies bedeutet, dass eine Applikation in einem TCL Skript oder, objektorientiert, mit incrTCL oder C++ programmiert werden kann. Objekte, die Zugriffe auf Systemdienste (z.B. Performer) benötigen, können jedoch nicht mit TCL, sondern nur mit C++ geschrieben werden. Dadurch kann der Entwickler von den Vorteilen beider Programmiersprachen profitieren. Zum einen lassen sich zeitkritische Applikationsteile in C++ erstellen, um die schnelle Ausführungszeit der Sprache auszunützen. Zum anderen können 3D-Userinterfaces aber auch sehr schnell durch die Skriptsprache Tcl programmiert werden, wobei es in Tcl ebenfalls

möglich ist, den Sprachumfang um Lightning-spezifische Befehle[5] zu erweitern. Um auch bei komplexen Programmen von weit mehr als 1000 Zeilen Code nicht die Übersicht zu verlieren, lässt sich außerdem eine Erweiterung der Sprache Tcl verwenden. Nach McLennan [mcl96] erweitert Incremental Tcl [incrTcl] die Sprache Tcl, um die objektorientierte Programmierung zu unterstützen. Dadurch wird es möglich, die Vorteile einer objektorientierten Sprache wie das Kapseln von Funktionalität in Objekte, das Definieren objektspezifischer Variablen, das Aufrufen von Methoden oder das Vererben von objektspezifischen Funktionen voll auszunützen. Somit werden einige Probleme, welche bei der Programmier ung von großen Applikationen in Skripten entstehen, gelöst.

3.1.5 Verwendung von Lightning auf Clustersystemen

Simulationen haben je nach ihrer Komplexität einen sehr hohen Leistungsbedarf an das Rechnersystem. Die Rechenlast der Simulation ergibt sich in der Regel aus der Komplexität des Rendervorgangs sowie weiteren Berechnungen für die in einer Simulation benötigten Zusatzwerte (z.B. Erzeugen generischer Flächen). In dieser Arbeit erzeugt hauptsächlich der Rendervorgang die Rechenlast. Dabei sind der Geometrieumfang (Anzahl der Polygone) der verwendeten Grafikobjekte sowie das verwendete Beleuchtungsmodell wesentliche Faktoren, die den Rechenaufwand im Rendersystem bestimmen. Zusätzlich benötigt der Umfang der verwendeten Texturen einen hohen Platzbedarf im Speicher des Computers. Durch die oben genannten sowie einige weitere Faktoren wie z.B. Berechnungstiefe des Anti-Aliasing und verwendete Shaderkonzepte oder Animationen werden für die Simulation besondere Anforderungen an den Computer gestellt.

In den letzten Jahren haben sich Clustersysteme aufgrund ihres Preises, gegenüber Multipiperechnern (Großrechner mit mehreren Grafikpipelines wie z.B. SGI Onyx) als vorteilhafter erwiesen. Der Einsatz von PC-Standardhardware in VR-Systemen basiert ebenfalls auf dem schnelleren Entwicklungstempo (vorangetrieben durch die Spieleentwicklung) der Haupt- und Grafikprozessoren.

Aus Programmierersicht besteht ein VR-Clustersystem aus Einzelrechnern, welche durch ein Netzwerk (z.B. Ethernet) miteinander verbunden sind. Jedes Einzelsystem besitzt seinen eigenen lokalen Speicher und eine lokale Grafikkarte, was sie wesentlich von Multipipesystemen unterscheidet, wobei jedoch bei beiden Systemen mehrere CPUs pro Einheit

---

[5] Nach der Konvention beginnen Lightningeigene Objekte immer mit einem angeführten lt. So zum Beispiel: ltroute, ltscript, ltvisobj, ltgetA, ltSetA… um nur einige zu nennen.

möglich sind. VR-Cluster sind somit verteilte Echtzeitrenderingsysteme mit besonderen Anforderungen.

Ein Clusterknoten pro Display (z.B. Projektor) erfordert die folgenden Vorraussetzungen:

- In jedem Frame müssen kohärente Szenenzustände auf allen Clusterknoten gewährleistet sein, da es sonst zu Laufzeitunterschieden zwischen den Knoten im Cluster kommt.
- Während des Rendervorgangs ist das anzuzeigende Bild noch unvollständig. Durch die Verwendung von Double Buffering wird dieser störende Effekt vor dem Betrachter verborgen. Dabei bedient man sich zweier Framebuffer, wobei der Frontbuffer immer angezeigt wird, während im Backbuffer das nächste Bild generiert wird. Ist die Generierung des Bildes abgeschlossen, führt die Grafikausgabe einen Buffer-Swap durch, bei welchem auf den Knoten zwischen Front- und Backbuffer umgeschaltet wird. Dieser Vorgang muss dabei auf allen Knoten synchron erfolgen.

Wesentliche Anforderungen, die an das Netzwerk gestellt werden, sind:

- Die Bandbreite muss, für alle Rechner im Cluster, zur gegenseitigen Synchronisation ein Datenpaket pro Frame gewährleisten.
- Geringe Latenzzeit des Netzwerkes zur Datensynchronisation ist ein wesentlicher Bestandteil, denn die verwendete Netzwerkbandbreite ist abhängig vom Verteilungsmodell der Applikation.

Das gewählte Verteilungsmodell hat einen erheblichen Einfluss auf die Konfiguration des Clusters. Dazu gibt es drei wesentliche Modelle.

Das von Lightning verwendete Modell der replizierten Simulation (s. Abbildung 12) generiert eine vollständige Kopie der Applikation sowie des Szenengraphen auf jedem Clusterknoten. Daher wird der Zustand der Simulation auf jedem Einzelrechner vollständig berechnet. Einzig die Eingabedaten des Tracking und der Interaktionsgeräte werden von dem Device Server verteilt, was zur Folge hat, dass die Netzwerklast so gering wie möglich gehalten wird. Allerdings liegen die Nachteile dieser Methode ebenso auf der Hand, denn zum einen sind ein Großteil der Berechnungen unnötig repliziert, da durch die Replikation die auszuführenden Berechnungen ohnehin auf allen Knoten die Selben sind. Zum anderen verlangt diese Art der Verteilung ein deterministisches Verhalten aller Komponenten im System, da sonst zum Beispiel bei der Verwendung von Zufallszahlen die einzelnen

Knoten im System nicht mehr dieselben Zustände besitzen. Daher müssen Zufallszahlen von einem Master berechnet und anschließend an die Slaves weitergegeben werden.

Manche Teile einer Applikation erfordern jedoch keine Ausführung auf allen Clusterknoten. Hierbei wären das Abspielen von Sound, was mittels eines auf dem Master gestarteten Soundserver geschieht, oder das Abfragen von Datenbanken, deren Inhalte anschließend vom Master an die Slaves weitergegeben werden, zu nennen.

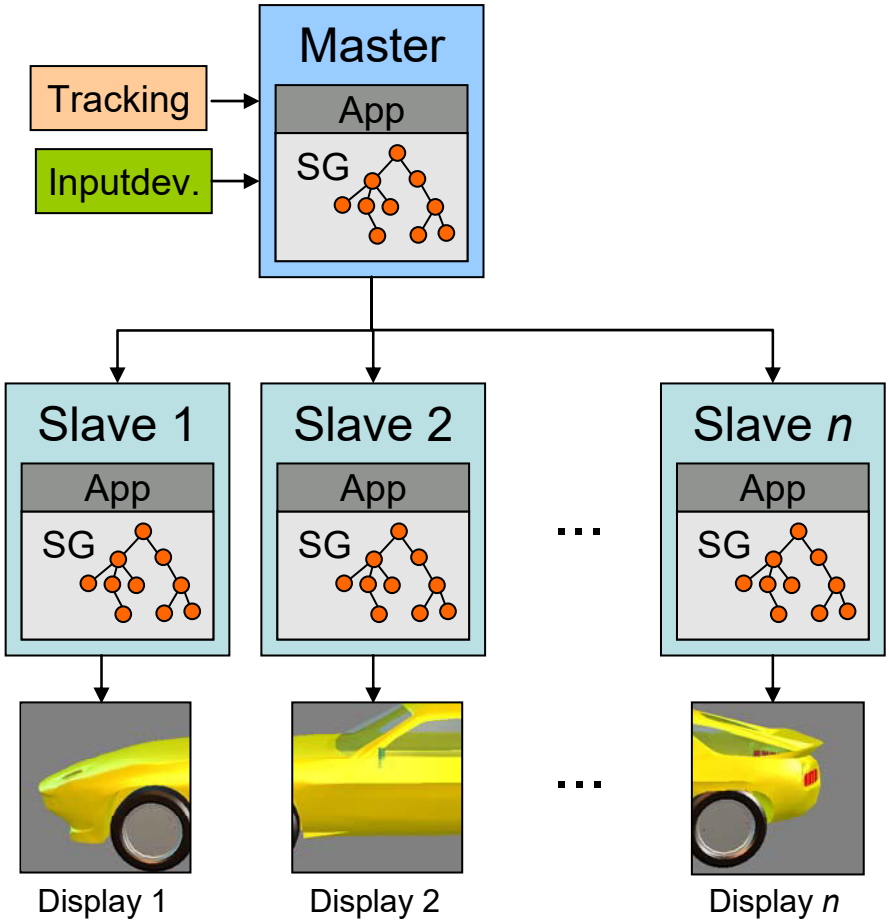


Abbildung 12: Veranschaulichte Darstellung der replizierten Simulation [bue03].

Eine andere Möglichkeit der Verteilung ist, dass die Applikation mit Szenengraph nur auf dem Masterknoten im Cluster läuft, wobei der dazugehörige Szenengraph auf allen Slaves repliziert ist. Hierbei werden Dateneingaben von Benutzerinteraktionen und des Trackings nur auf diesem Rechner verarbeitet. Die Updates, welche den Szenengraph tangieren, wie Änderungen der Transformationen, Hierarchien oder Attribute von visuellen Objekten (Farbe, Transparenz…) werden an die Slaves weitergegeben. Dadurch wird eine geringere Anzahl der zu replizierenden Tasks bei einer gering höheren Netzlast erreicht. Diese Methode findet zum Beispiel bei Open SG (Open Source Scene Graph) seine Verwendung.

Abschließend soll der Distributed-Immediate-Mode genannt werden, bei welchem es (außer den Displaylisten) keinerlei Replikation der Anwendung oder des Szenengraphen gibt. Änderungen, welche die Slaves betreffen, werden über komprimierte OpenGL Befehle an die Slaves delegiert. Dadurch ist diese Methode weitgehend von der Applikation unabhängig, wobei die volle Ausnutzung der Grafikressourcen der Slaves nur mittels Caching der Geometrien (Displaylisten) möglich ist. Die Netzwerklast ist daher direkt von der Komplexität der Szene abhängig.

3.2 Weitere verwendete Softwarekomponenten

Die Computergrafik bedient sich einer Vielfalt an Werkzeugen, um 3D-Geometrien zu erzeugen. Diese sollten, um den Grad der Immersion zu steigern, so realistisch wie möglich erscheinen. Ansprechende Modelle zu generieren ist nur durch den korrekten Einsatz dieser Tools und der Beachtung von gestalterischen Grundlagen möglich. In den folgenden Kapiteln wird diskutiert, worauf bei der Erstellung von 3D-Modellen für den Einsatz in virtuellen Umgebungen geachtet werden muss und welchen Anforderungen und Rahmenbedingungen diese unterliegen.

Tabelle 1: Verwendete Softwarekomponenten zur Modellerstellung und Texturierung.

Verwendete Software	Betriebssystem	Einsatzgebiet der Software
Discreet; 3D Studio Max	Windows XP	Modellierung und Texturierung der Geometrie und Export in VRML.
Adobe; Photoshop	Windows XP	Erzeugen und Bearbeiten von Bildern zur Texturierung des Modells.
Silicon Graphics, Inc.; Performer	SUSE Linux	Formatkonvertierung von VRML nach PFB.
Silicon Graphics, Inc.; SMGEN	IRIX	Erzeugen von Environment Maps.
Parallel Graphics; Cortona Viewer	Windows XP	VRML Plug-in für den MS Internet Explorer.
GIMP	SUSE Linux	Konvertierung von Bildformaten.
XnView	Windows XP SUSE Linux	Konvertierung von Bildformaten.

3.2.1 Modellierprogramme zur Erstellung von echtzeitfähigen Modellen

Mittlerweile ist man durch die Flut qualitativ hochwertiger Bilder, welche mit Hilfe der Technik von Computergrafik entstanden sind, äußerst verwöhnt. Beispiele hierfür finden sich in der Werbung in Zeitschriften und Fernsehen, in Computerspielen und Kinofilmen. Die Animationen und Grafiken strahlen eine solche Detailvielfalt und Vollkommenheit aus, dass sie mittlerweile oft schon schwer von „echten Bildern“ zu unterscheiden sind. Dem Vergleich mit derartigen Grafiken müssen sich die Anwendungen der Virtual Reality häufig stellen. Hierbei ist jedoch zu beachten, dass die Generierung (hier das Rendern) eines einzelnen Bildes bei statischen Bildern oder animierten Filmen selbst mit der Hilfe eines leistungsstarken Rechners nur mit außerordentlichem Zeitaufwand möglich ist. Renderzeiten von mehreren Stunden für ein einzelnes Bild sind dabei keine Seltenheit. Jedoch fordern Simulationen in Virtual Reality eine Framerate von mindestens 15 FPS[6]. Das bedeutet, dass pro Einzelbild eine maximale Zeit von ca. 0.0667 Sekunden für die Berechnung in Anspruch genommen werden darf. Die Notwendigkeit der Mindestframerate verdeutlicht Sherman [she03] wie folgt:

> *„Als untere Grenze werden Raten von 15 Hz noch als akzeptabel betrachtet, da Wiederholraten unterhalb der 15 Hz bei einigen Betrachtern eine Übelkeit mit Schwindelzuständen [7] auslösen können“.*

Die erreichbare Framerate ist von vielen Faktoren abhängig, wobei die verwendete Hardware wie zum Beispiel die Leistung der CPU und GPU, die Kapazität des Arbeits- und Texturspeichers oder die Taktrate von Systembus und Speicher den maximalen Rahmen der Darstellungsmöglichkeit angibt. Um eine optimale Auslastung der Systemressourcen zu gewährleisten, ist bei der Modellierung von architektonischen Modellen besonders auf den geometrischen Umfang der Modelle zu achten, was zur Folge hat, dass die Anzahl der verwendeten Punkte, Kanten und Flächen, welche das 3D-Objekt definieren, so gering wie möglich gehalten wird. Die davon abgeleitete Aussage, je weniger Polygone, desto performanter die Darstellung, bedeutet allerdings einen Detailverlust des Modells und steht somit in Konfrontation mit dem oben genannten Anspruch der realistischen Darstellung.

Heutige CAD Programme zur Erstellung von Computeranimationen (z.B. 3D Studio Max, Maya, Rhino, Cinema4D, etc.) bieten alle Möglichkeiten, die ein Modellierer benötigt, um ein ansprechendes und echtzeitfähiges Modell zu erstellen. Hierfür ist es notwendig, die

---

6 Frames per second bezeichnet die Anzahl der Bilder, die pro Sekunde dargestellt werden.

7 Auch als „Cyber Sickness“ bekannt.

Realität, welche wiedergegeben werden soll, zu abstrahieren und nur die bedeutsamen Details in das Modell zu integrieren. Nach Brugger [bru95] bietet die Computergrafik als unglaublich flexibles Medium die Gelegenheit, auch in höheren Abstraktionsebenen sehr aussagekräftige und realitätsnahe Eindrücke zu erzeugen. Dadurch hat der Computergrafiker die Möglichkeit, den Grad des Realismus, mit dem er die Wirklichkeit wiedergibt, selbst zu bestimmen. Dies erbringt besonders bei Architekturmodellen mit vielen Rundbögen einen Vorteil, da es somit möglich ist, die Anzahl der Polygone, welche z.B. einen Rundbogen darstellen, im Modell so gering wie nötig zu bemessen, um das Dargestellte noch unmissverständlich wiederzugeben. Die wichtigste Voraussetzung für eine gelungene Visualisierung auf einer bestimmten Realitätsstufe bzw. Abstraktionsebene ist hierbei, dass alle Teile, welche die Gesamtszene ausmachen, auf der gleichen Stufe stehen, also denselben Grad an Realismus aufweisen. Es muss ein homogener, durchgängiger Bildeindruck entstehen, denn bereits ein einzelnes aus der Reihe fallendes Detail wie ein zu grob modelliertes Objekt kann die Ausgewogenheit zerstören.

Um den insgesamt notwendigen Geometrieaufwand abschätzen zu können, muss der Modellierer sich zunächst ein genaues Bild des zu errichtenden Bauwerks machen. Je höher der Detailgrad eines Bauwerks, zum Beispiel einer gotischen Kirche, desto mehr muss er sich darauf konzentrieren, die wesentlichen Teile des Gebäudes wiederzugeben und die zu Verfügung stehende Polygonanzahl so effektiv wie möglich einzusetzen. Somit kann der Grafiker durch das richtige Verhältnis von Geometrieumfang und einer „Harmonie des Ganzen“ einen für den Betrachter realistischen Eindruck erzeugen. Die Renderzeiten werden dabei so gering wie möglich gehalten, wodurch die Darstellungsqualität in der Simulationsumgebung für den Betrachter steigt.

3.2.2 Digitale Bildverarbeitung zur Texturerstellung

Eine detailreichere Darstellung ohne unmittelbar steigende Geometriekomplexität wird durch die Verwendung von Texturen auf den 3D-Objekten erreicht. Damit ist es möglich, Bilder auf die Flächen der Geometrie eines Modells zu legen (Mapping), um einen realistischeren Eindruck zu erwecken [hec86]. Die dafür verwendeten Bilder müssen sorgfältig ausgewählt und bearbeitet werden, da in der Computergrafik auch an dieser Stelle besondere Vorraussetzungen zu erfüllen sind. Wesentlich dabei ist, dass der Charakter des Modells durch die Verwendung von ausgewählten Texturen erst richtig zur Geltung kommen kann. Hierbei sollte man sich so nah wie möglich an die Eigenschaften des Originals halten, wodurch ein sehr hoher Wiedererkennungswert mit dem Vorbild erreicht wird. Nach Brugger können vorgefertigte Materialbibliotheken diesem Anspruch unmöglich Rechnung

tragen. Die Farben von Materialien in Standard-Bibliotheken sind in den meisten Fällen ein „Sammelsurium" aus unterschiedlichen Quellen. So trifft nicht nur eine Vielzahl von Geschmäckern, Stimmungen und Vorlieben aufeinander. Die Materialfarben sind zudem aus verschiedenen Anforderungen und unter abweichenden Zielvorgaben entstanden. Außerdem wurden sie für die verschiedensten Gelegenheiten und Fachrichtungen entworfen.

Des Weiteren ist darauf zu achten, dass alle verwendeten Materialien und Texturen einen gemeinsamen Farbcharakter beinhalten. Die prinzipiellen Grundvoraussetzungen für harmonische Farbkompositionen lassen sich nach Brugger aus den Qualitätsmerkmalen (ästhetischen Unterscheidungsmerkmalen) der Farben ableiten. In der 3D-Computergrafik werden Farbton, Helligkeit und Sättigung als Qualitätsmerkmale unterschieden. Wenn Farben die gleichen ästhetischen Merkmale aufweisen, entstehen Bindungen zwischen ihnen, die zu sichtbaren Gemeinsamkeiten (Harmonien) führen. Zusätzlich ist auf das eigentliche Motiv der Textur selbst zu achten. So führen Schattenwürfe und eine unrealistische Perspektive des auf der Textur dargestellten Bildes in der Simulation meist zu einem verwirrenden Eindruck, da der Betrachter nicht an eine bestimmte Perspektive gebunden ist, sondern sich komplett frei bewegen kann. Eine vom Standort und Blickwinkel abhängige Textur ist somit zu vermeiden.

Adobe Photoshop ist die wohl bekannteste professionelle Bildbearbeitungssoftware auf dem Markt. Mit deren Hilfe ist es möglich, Fotografien und Scans so zu bearbeiten, dass daraus hochwertige Texturen für das 3D-Modell entstehen. Farbkorrekturen, Alphakanäle, Kontraste, Sättigungen, Schärfe/Unschärfe und Filter ermöglichen es, auch aus bei unterschiedlichen Lichtverhältnissen aufgenommenen Bildern Texturen zu erzeugen, die in Einklang zueinander stehen. Des Weiteren ist es mit den modernen Bildbearbeitungsprogrammen möglich, Texturen zu generieren, welche an jeder Kante nahtlos aneinanderpassen. Woraus diese Anforderung resultiert, wird in Kapitel 4 näher erläutert.

3.2.3 Export in andere Datenformate (VRML)

Mittlerweile gibt es kaum noch 3D-Konstruktionsprogramme, die nicht nach VRML exportieren. Damit ist eine nahezu vollständige Unterstützung bei den professionellen 3D-Erstellungstools gegeben, welche auf den Märkten CAD, mechanische Konstruktion und Architektur von ihren Kunden verwendet werden. Beispiele hierfür sind CATIA, Microstation, AUTOCAD, 3D Studio Max und Maya. Das Magazin iX des Heise Verlags [dec03] erklärt die weite Verbreitung des Datenaustauschformates folgendermaßen:

„Im Vergleich zu anderen 3D-Formaten, insbesondere gegenüber proprietären Formaten von VR-Softwareherstellern verfügt die Virtual Reality Modeling Language über folgende Vorteile:

- *Exportfilter gängiger Softwarelösungen im CAD- und Modellierungsbereich sind nutzbar.*
- *Softwarepakete wie 3D Studio Max von Discreet und Cinema 4D von Maxon beinhalten VRML-Modellierwerkzeuge.*
- *Einige Hersteller bieten spezielle VRML-Programmierwerkzeuge an (beispielsweise VRML-Pad von Parallelgraphics).*
- *Auch VRML-Code-Optimierwerkzeuge sind verfügbar (etwa Chisel von Trapezium).*
- *Texturen und Sounds erhöhen den Immersionseffekt.*
- *VRML-Sensoren erlauben eine umfangreiche Interaktion.*
- *VRML-Interpolatoren sorgen für viel Dynamik.*
- *Die GeoVRML-Erweiterungen ermöglichen die Darstellung von Geologiedaten.*
- *Javascript erlaubt eine beliebige Objektmodellierung (Logik, Abhängigkeiten, Kinematik).“*

In dieser Arbeit kommt VRML als Datenaustauschformat zum Einsatz, da 3D Studio Max einen guten Plug-in für den Expot von VRML bereithält. Lightning beinhaltet Loader für die Geometrie vieler verschiedener Formate, darunter auch einen VRML-Loader, basierend auf libVRML97 von Chris Morley [mor03]. Jedoch werden nicht alle Features des VRML Standards verwendet, denn um sehr komplexe Logik zu programmieren, eignet sich die [incrTcl] Programmierschnittstelle von Lightning um ein Vielfaches besser, als diese Funktionalität in VRML zu beschreiben. Als besonderer Vorteil ist natürlich das ASCII Format von VRML zu erwähnen. Kleine Änderungen an Materialeigenschaft oder Position eines Objekts lassen sich mit Hilfe eines Texteditors sehr schnell und unkompliziert vornehmen. Ein großer Nachteil ist jedoch, dass VRML keine Unterstützung von Multitexturierung oder Reflection Mapping etc. bietet. Besitzt ein Objekt z.B. mehrere Texturen, was mit 3D Studio Max keinerlei Probleme hervorruft, so wird diese Information beim Export des Modells in VRML verworfen.

3.3 SQLite, eine Datenbank mit Tcl API

Für die Ablage von Informationen bietet es sich auch bei Applikationen im Bereich Virtual Reality an, diese Daten in eine externe Datenbank zu speichern. Der große Vorteil dabei ist, dass sich die Informationen in einer Datenbank besser verwalten und erweitern lassen, als wenn diese direkt in der Applikation abgelegt sind.

Da für die in dieser Arbeit bereitzustellenden Informationen eine kleine Datenbank völlig ausreicht, fand die frei verfügbare SQL-Datenbank SQLite hier ihren Einsatz. Diese dient lediglich der dauerhaften Ablage sowie der schnellen Abfrage der gewünschten Informationen. Dafür bietet SQLite eine direkte Schnittstelle (Interface) zur Programmiersprache TCL an [hip04], wodurch SQL-Abfragen auf diese Datenbank leicht in die Applikation integriert werden können. Zusätzlich steht zur Dateneingabe und Pflege ein Datenbankbrowser zu Verfügung, mit welchem Änderungen sehr komfortabel in den Datenbanktabellen vorgenommen werden können.

4 Erstellung von echtzeitfähigen Modellen

Als Modellierungsgrundlage für das virtuelle Abbild der Stiftskirche Stuttgart dient ein maßstabsgerechter Grundrissplan des Bauwerks aus dem Jahr 1992. Erweiterte Informationen über die Dimension des Mittelschiffs, der Seitenschiffe und des Chors sind in dem von Oliver Auge erschienenen Buch „Kleine Geschichte der Stuttgarter Stiftskirche“ zu finden [aug01]. Zusätzlich wurden bei mehreren Begehungen Aufnahmen mit einer Digitalkamera von außen und im Inneren der Kirche gemacht. Anhand dieser Aufnahmen können viele Details des Bauwerks, welche aus dem Grundriss nicht ersichtlich sind, wahrgenommen und in das Modell übertragen werden. Außerdem dienen sie der Generierung von Texturen. Weitere Bilder, welche Anhaltspunkte und Vorgaben für die Modellierung liefern, sind in dem Buch „Die Stiftskirche in Stuttgart“ zu sehen [sor].

In den folgenden Kapiteln wird ausführlich erläutert, wie unter Berücksichtigung der in Kapitel 3.2.1 beschriebenen Ansprüche an ein echtzeitfähiges 3D-Modell, die Geometrie der virtuellen Stiftskirche Stuttgart, entsteht. Zusätzlich werden die dabei verwendeten Funktionen des Modellierwerkzeugs 3D Studio Max anhand ausgewählter Beispiele erklärt.

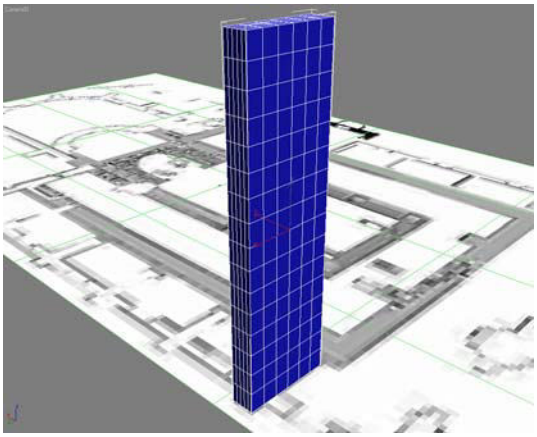
4.1 Mesh und Poly Modellierung

Wie der Name Mesh schon sagt, gleichen mit dieser Methode erstellte Modelle einem Netz. Dieses Netz besteht aus vier Subobjekten.

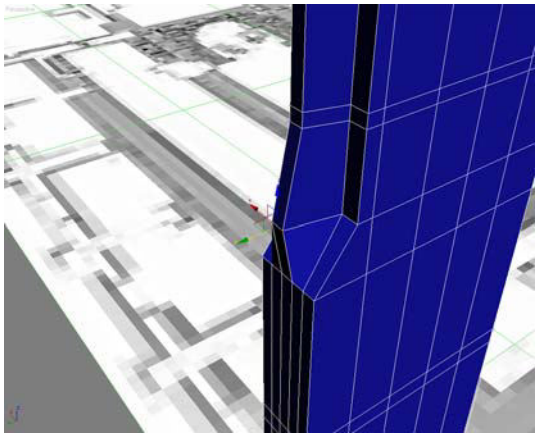
- **Scheitelpunkt** (Vertex) ist der Eckpunkt eines Polygons.
- **Kanten** (Edges) sind Verbindungen zwischen zwei Vertices.
- **Flächen** (Faces) ergeben sich aus 3 Vertices, welche durch Kanten verbunden sind.
- **Elemente** sind mehrere zusammengehörige Flächen eines Objekts.

Poly-Objekte bestehen aus Flächen mit mehr als drei Eckpunkten und sind somit eine Sonderform der Mesh-Objekte. Vorteile der Poly-Objekte zeigen sich bei der Anwendung von Modifikatoren wie Abkanten von Edges oder Zerschneiden von Körpern. Zusätzlich besitzen sie das Subobjekt Border, welchem alle Kanten, die nur an eine Fläche grenzen, zugehören. Bei der Erstellung von Mesh- oder Polygonmodellen wird meistens von einem Grundkörper (parametrisierbares Objekt) wie Kugel, Quader, Kegel oder Torus ausgegangen, dem ein gewünschter Geometrieumfang vorgegeben wird. Nach der Konvertierung

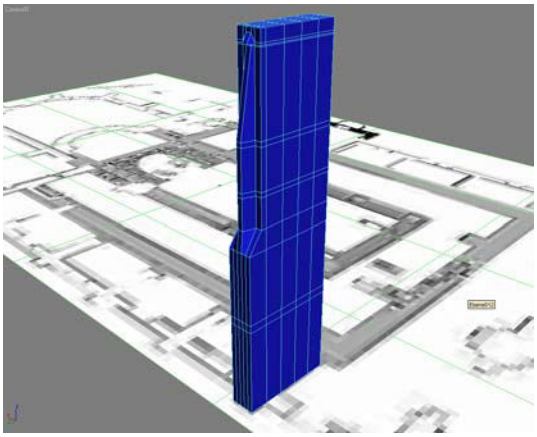
des Grundkörpers in ein Mesh- oder Poly-Objekt kann dieses mittels Modifikation der einzelnen Punkte in die gewünschte Form gebracht werden, wobei mit Poly-Objekten ein wesentlich intuitiveres Arbeiten möglich ist. In der folgenden Abbildung wird das Prinzip am Beispiel einer Außenwand der Stiftskirche veranschaulicht.



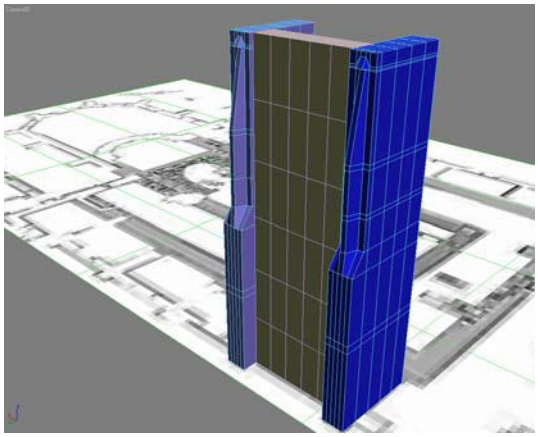
a) Grundkörper definieren



b) Verändern der Geometrie



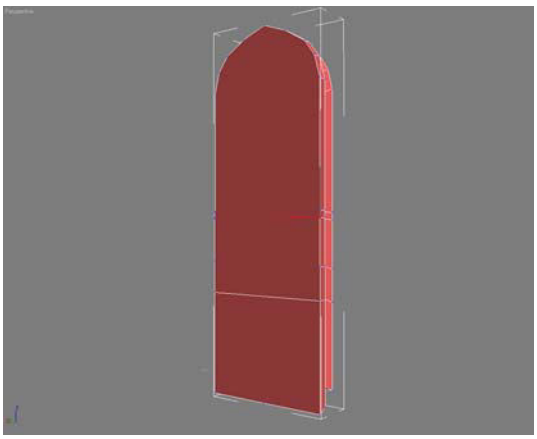
c) Fertiges Objekt



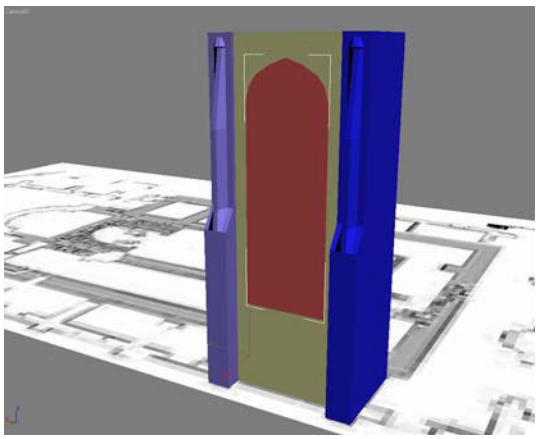
d) Erweitern mit zusätzlichen Objekten

Abbildung 13: Vom Grundkörper zum gewünschten Poly-Objekt.

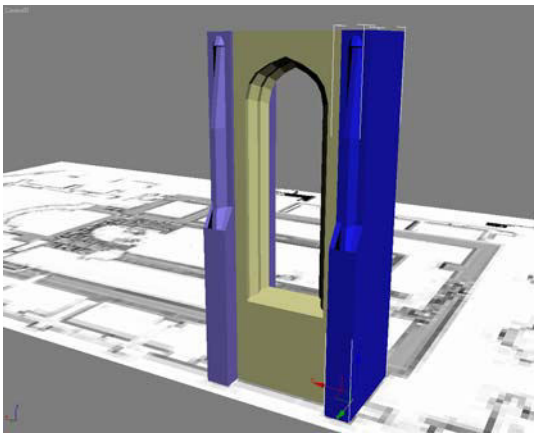
Um nun auch Öffnungen für Türen und Fenster in die Geometrie zu integrieren, werden boolesche Operatoren verwendet. Dabei wird ein Objekt zu einem anderen addiert, davon subtrahiert oder die Schnittmenge aus beiden errechnet. In dem in Abbildung 14 dargestellten Bild wird also eine Form, welche ein Fenster darstellt, aus der Wand herausgeschnitten. Um eine realistische Darstellung der Fenster zu gewährleisten, müssen die Rundungen und Verjüngungen dem gotischen Original entsprechen. Hierbei muss jedoch besonders auf den geometrischen Umfang des Hilfsobjekts geachtet werden, da sich dessen Geometrie beim Ausschneiden auf das andere Objekt überträgt.



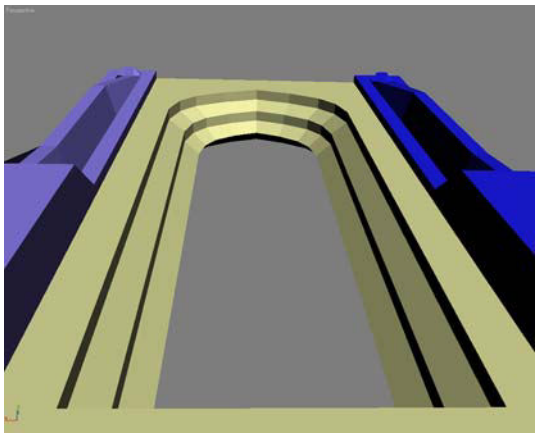
a) Grundform erstellen



b) Hilfskörper positionieren



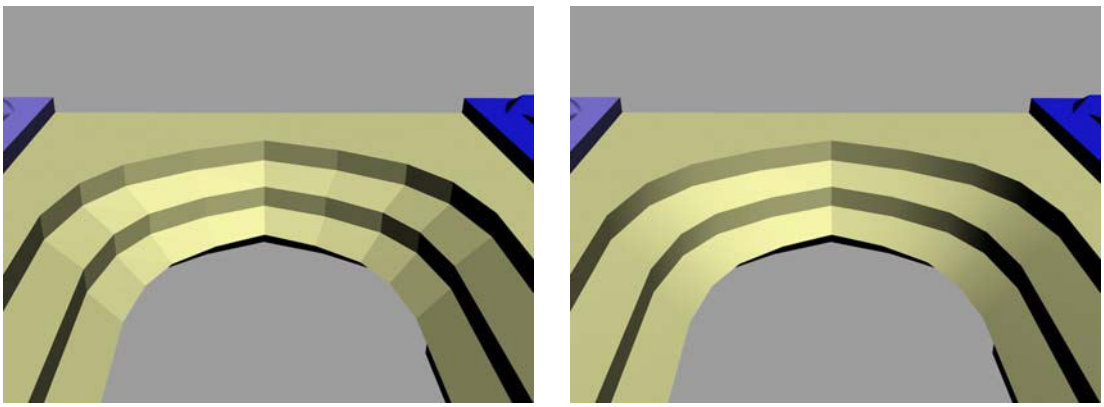
b) Nach der booleschen Operation



c) Fenster im Detail

Abbildung 14: Wirkungsweise von booleschen Operationen.

Auffällig ist der Bogen in Abbildung 14 c). Er weist kantige, facettierte Stellen an den Flächen auf, da dort sehr wenig Polygone vorhanden sind, um eine Rundung zu beschreiben. Jede dieser Flächen besitzt einen Normalenvektor, der orthogonal auf der Fläche steht. Die benachbarten Flächen sind in ihrer Ausrichtung aber leicht voneinander weggedreht, wodurch dieser eckige Effekt in Abhängigkeit vom Winkel des Lichteinfalls entsteht. Hier bietet 3D Studio Max die Möglichkeit Flächennormalen, welche in einem bestimmten Winkel zueinander stehen, zu interpolieren. Die flach schattierten Flächen, deren Normalen innerhalb des angegebenen Winkels zueinander stehen, werden in die Berechnung mit einbezogen und somit glatt gezeichnet, ohne steigenden Geometrieumfang des Objekts. Anhand eines Ausschnitts des Rundbogens soll dies in Abbildung 15 gezeigt werden.



a) Facettierte Flächen *b) Glatte Flächen*

Abbildung 15: Smoothing durch Normaleninterpolation.

Das hier beschriebene Beispiel ist ein Segment der Nordwand der Stiftskirche, welche aus sechs identischen Seitenkapellen besteht. Daher ist es möglich, das Segment zu kopieren, um die Wandseite zu vervollständigen. 3D Studio Max stellt hierfür drei unterschiedliche Duplizierungsmethoden bereit. Da das Kopieren nur eine Art des Duplizierens ist, wird dies in 3D Studio Max als Klonen bezeichnet. Das Vervielfältigen geschieht grundsätzlich bei der Transformation von Objekten. Zusätzlich zu den Transformationsmethoden Bewegen, Skalieren und Rotieren gibt es aber auch die Möglichkeit, durch Anordnung, Spiegeln und Momentaufnahme zu duplizieren [vel02].

- **Kopien:** Das einfache Kopieren ist die bekannteste Art, Objekte zu duplizieren. Dabei wird ein genaues Abbild des Originals erstellt. Der neu entstandene Körper ist ein eigenständiges Objekt, welches unabhängig vom Ursprungsobjekt verändert werden kann. Kopien werden dort verwendet, wo ein völlig neuer Körper erstellt wird, der die Grundform eines bereits bestehenden Objekts besitzt.

- **Instanzen:** Durch Instanzierung können Objekte unbegrenzt vervielfältigt werden. Daher tauchen Instanzen vor allem dort auf, wo viele identische Objekte wie Bäume, Zaunlatten usw. benötigt werden. Änderungen an der Geometrie einer Instanz übertragen sich automatisch auf alle anderen Objekte. Somit sind Instanzen keine eigenständigen Objekte.

- **Referenzen:** Referenzen sind (gewissermaßen) Instanzen, die nur in eine Richtung funktionieren. Referenzobjekte basieren ebenso wie Instanzen auf dem Ursprungsobjekt und können über eigene Modifikatoren verfügen. Alle Änderungen am Ursprungsobjekt werden auf dessen Referenzen übertragen; Änderungen an einer

Referenz werden jedoch nicht vom Ursprungsobjekt und den anderen Referenzen übernommen. Referenzen sind daher konzipiert, um ein Objekt, welches einem anderen in den Grundzügen gleicht, unabhängig davon modifizieren zu können.

Bei der Vervollständigung der Nordwand bietet es sich an, das Segment über Referenzen zu kopieren, da sich nachträglich vorgenommene Änderungen am Vaterobjekt somit automatisch auf die einzelnen Objekte übertragen und dem Modellierer dadurch ein erheblicher Zeitaufwand erspart werden kann.

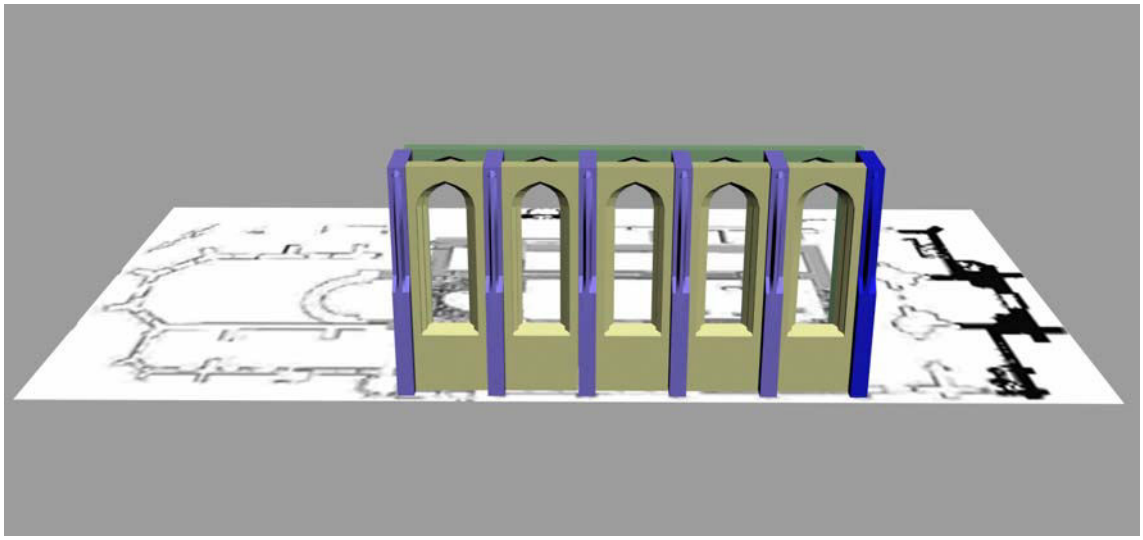


Abbildung 16: Resultat der Referenzierung.

Kopien, Instanzen und Referenzen lassen sich aber nicht nur bei geometrischen Objekten, sondern auch bei Materialien, Modifikatoren, Gruppen, Kameras, Lichtern usw. anwenden.

4.2 Texturierung

Da die Nordfassade des Bauwerks aus Sandstein besteht, wird schließlich eine entsprechende Textur benötigt, um diesen Eindruck realistisch vermitteln zu können. Aus unterschiedlichen Bildern, welche mit einer Digitalkamera von der Stiftskirche aufgenommen wurden (vgl. Abbildung 17), ist es möglich, eine Textur der Sandsteinmauer zu generieren.

Abbildung 17: Fotos der Sandsteinfassade der Stiftskirche.

Die Wand hat eine Länge von ca. 15 und eine Höhe von ca. 7 Metern. Um diese mit Inhalt zu versehen, werden Texturen verwendet, welche die Eigenschaft besitzen, sich nahtlos aneinander reihen zu lassen (tiled textures). Der Vorteil dieser Methode ist, dass das Bild in einer sehr hohen Auflösung verwendet werden kann und trotzdem nicht zu viel Platz im Texturspeicher benötigt. Dazu werden aus den Bildern von Abbildung 17 einzelne Steinblöcke mittels Photoshop herausgetrennt und in ihrem Kontrast und ihrer Farbeigenschaft aufeinander abgestimmt (Abbildung 18).

Die einzelnen Fragmente lassen sich nun in eine große Textur zusammenfügen. Hierbei werden mehrfach kopierte Teile unterschiedlich skaliert, damit die Wiederholung der einzelnen Steine innerhalb des Bildes für das Auge nicht ersichtlich ist. Ebenso bewirkt auch eine leichte Farbabtönung einzelner Steine, dass auftretende Muster vom Auge nicht sofort als Kopie entlarvt werden (s. Abbildung 19). Die Textur hat eine Auflösung von 1024 x 1024 Bildpunkten. Weitere mögliche Auflösungen für Texturen sind z.B. 512 x 1024 oder 256 x 512 Bildpunkte. Die Größe des Bildes auf Zweierpotenzen zu fixieren ist insofern notwendig, da der Texturspeicher in den Grafikkarten so optimal ausgenützt werden kann.

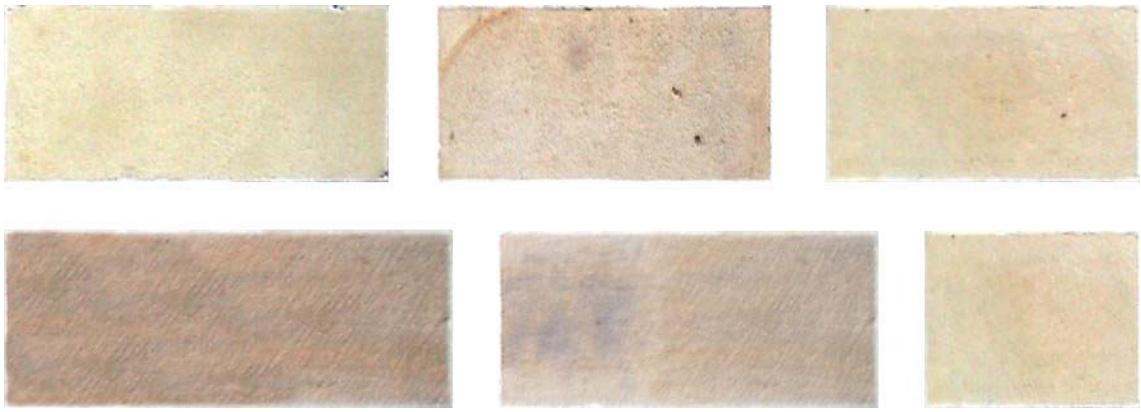


Abbildung 18: Einzelne, angepasste Texturstücke.

Abbildung 19: Fertige Textur der Sandsteinwand.

4.3 Material und Mapping in 3D Studio Max

Um die Textur nun auf die Geometrie zu übertragen, ist es notwendig, ein Material zu generieren. Da Materialien und die Beleuchtung für einen realistischen Eindruck hauptverantwortlich sind, bietet 3D Studio Max einen Materialeditor mit einer Fülle von Einstellungsmöglichkeiten an. Im Folgenden werden einige grundlegende Eigenschaften, welche notwendig sind, um ein realistisches Material zu erzeugen, näher erläutert.

- Grundlegende Materialeigenschaften einer Fläche sind eine diffuse Farbe (Streufarbe) sowie weitere Farbanteile wie der emissive, ambiente und speculare Anteil. Diese haben bei variierenden Lichtverhältnissen unterschiedliche Wirkungsweisen auf den Farbeindruck der Fläche. Im Dunkeln beispielsweise ist der emissive der einzig sichtbare Farbanteil des Materials. Wird es jedoch von einem Licht angestrahlt, kommt der speculare Anteil in Form eines unterschiedlichen Glanzverhaltens zur Geltung. Eine zusätzliche Eigenschaft von Materialien ist die Transparenz.

- In 3D Studio Max bilden Shader Zusatzkomponenten eines Materials und werden durch das so genannte Mapping ergänzt. Das Mapping stellt den Unregelmäßigkeitsfaktor des Materials dar, was bedeutet, dass jede Eigenschaft des Shaders durch ein Mapping variiert werden kann. Auf dem diffusen Kanal kann somit eine Textur angegeben werden. Weitere Eigenschaften wie Reflexion, Transparenz, Self-Illumination (Selbstleuchten), Bump (Reliefmuster wie Steinstruktur oder Kratzer), Refraktion (Brechung) usw. werden mit Bitmaps oder parametrisch erzeugten Mustern verändert.

Des Weiteren beinhaltet 3D Studio Max zusätzliche Möglichkeiten, um spezielle Materialkombinationen und Effekte wie Glow (Glühen) zu erzeugen [vel02]. Da mit VRML allerdings nur die im ersten Punkt beschriebenen Eigenschaften sowie die Textur mit den zugehörigen Texturkoordinaten exportiert werden können, wird an dieser Stelle auf eine nähere Betrachtung von Shadern verzichtet. Eine ausführliche Beschreibung von aktuellen Shadermodellen findet sich im „CG Toolkit Users Manual“ von NVIDIA [nvi04].

In Lightning werden besondere Materialeigenschaften wie zum Beispiel Spiegelungen mit Hilfe von in Lightning definierten Materialien erzeugt und den gewünschten Objekten zugewiesen.

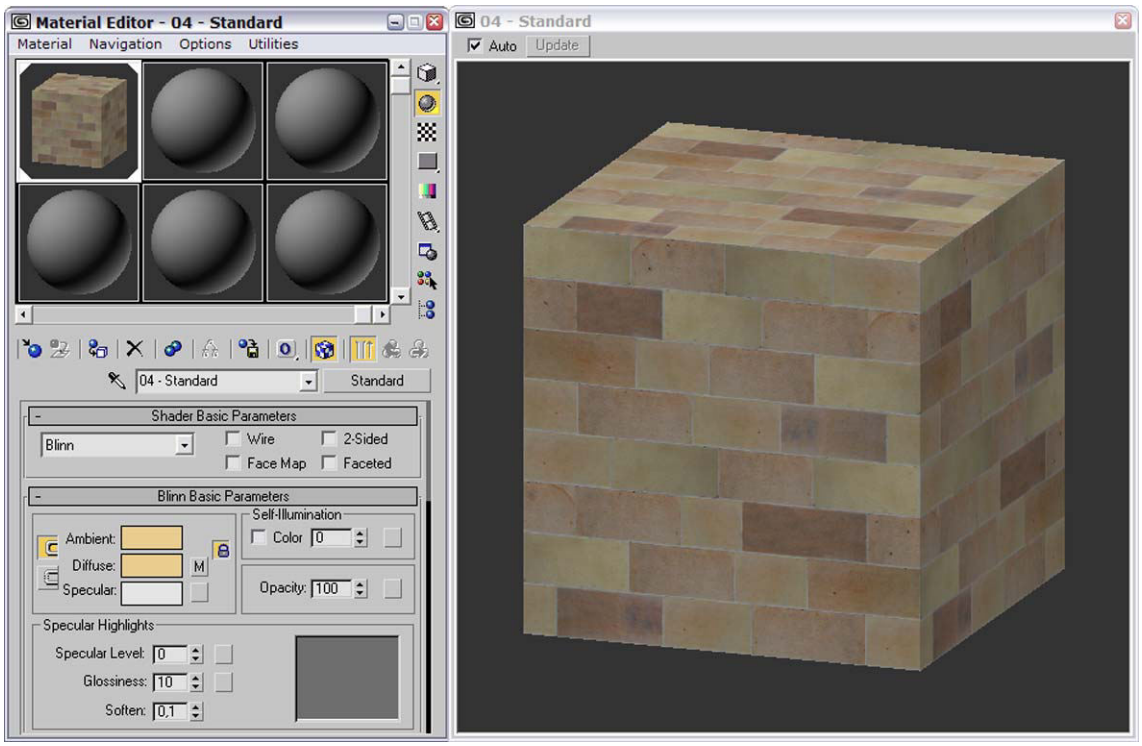


Abbildung 20: Materialeditor mit dem erstellten Material der Sandsteinmauer.

Nachdem das gewünschte Material erstellt wurde (s. Abbildung 20), lässt sich dieses auf die Geometrie der Nordwand mappen. Dabei wird eine Position definiert, an die das Mapping aufgebracht werden soll. Zu beachten ist hierbei, dass das Mapping nicht auf eine Fläche gelegt, sondern von einem bestimmten Punkt im Raum auf diese projiziert wird. Dazu werden, in Abhängigkeit von der Form des zu texturierenden Objekts, unterschiedliche Projektionsarten verwendet, welche sich mit einem Gizmo (Hilfsobjekt) verändern lassen. Im folgenden Abschnitt wird beschrieben, wie diese sich verhalten.

- **Planar:** Bei der flachen Projektion wird im rechten Winkel zum Gizmo abgestrahlt. Auf Flächen, welche orthogonal zur Projektionsrichtung stehen, wird ohne perspektivische Verzerrung projiziert.
- **Quader (box):** Auf jede Seite des Quaders wird das Mapping planar projiziert.
- **Zylindrisch (cylindrical):** Die Mittelachse eines Zylinders ist der Ausgangspunkt der Projektion. Die Projektionsstrahlen verlaufen dabei rechtwinklig zur Mittelachse, womit das Mapping auf das zu texturierende Objekte aufgetragen wird.
- **Kugelförmig (spherical):** Das Mapping wird aus dem Mittelpunkt der Kugel in alle Richtungen projiziert und somit kugelförmig auf das Objekt aufgetragen.
- **Fläche (face):** Das Mapping wird auf jede Fläche der Geometrie planar projiziert.

Aufgrund der rechtwinkligen Geometrie bietet es sich in diesem Fall an, eine kubische Projektion der Textur anzuwenden. Der Gizmo hat dabei eine Kantenlänge von 2.5 Metern. Bei einem Darstellungsmaßstab von 1:1 hat eine Steinreihe auf der Textur somit eine Höhe von ca. 30 cm, was dem Original der Stiftskirche annähernd entspricht. Durch die Kachelung der Textur wird diese nun auf dem gesamten Objekt wiederholt, was den Eindruck einer gemauerten Wand erzeugt.

Abbildung 21: Mit Textur versehene Nordwand der Stiftskirche.

4.4 Modellierung mit NURBS-Flächen

Auf die in Kapitel 4.1 beschriebene Art und Weise können viele Teile der Kirche wie zum Beispiel die Wände im Chor, der Fußboden, die Fenster oder die Südwand modelliert werden. Für die Erzeugung von Geometrien mit vielen feinen geometrischen Rundungen ist diese Methode jedoch nur schwer einsetzbar. Um die geschwungene Decke im Kirchenchor zu modellieren, wurde aus diesem Grund auf die Modellierung mit NURBS[8]-Flächen [til83] [pie91] zurückgegriffen.

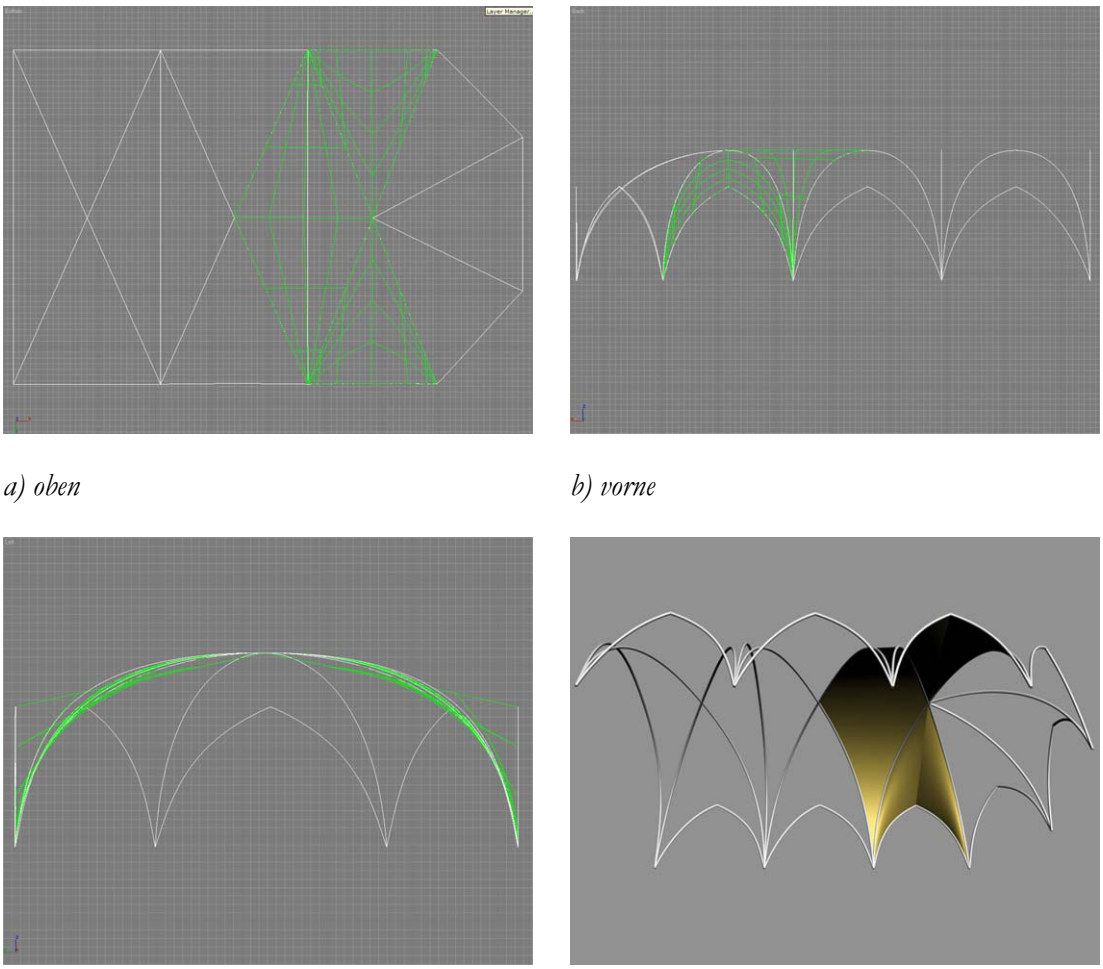
Zur Darstellung von Freiformkurven, -flächen oder -volumina sind NURBS bei der geometrischen Modellierung mittlerweile eine etablierte Methode der Modellerstellung. NURBS beschreiben die 3D-Objekte in analytischer Form. Um eine grafische Darstellung zu ermöglichen, müssen Objekte durch Polygone approximiert werden (Triangulierung). Die Genauigkeit der Approximation kann dabei vorgegeben werden, womit sich auf die Anzahl der zur Darstellung verwendeten Polygone Einfluss nehmen lässt.

Da die Modellerstellung mit NURBS sehr kompliziert ist und die gewölbte Decke des Chors eine komplexe geometrische Form darstellt, ist eine durchdachte Vorgehensweise bei der Modellierung erforderlich. In 3D Studio Max lassen sich Kurven als Grundlinien der Geometrie mit rationalen B-Splines zeichnen, die sich mit Hilfe ihrer Kontrollpunkte und Gewichte intuitiv formen lassen. Zwischen solchen Kurven können nun NURBS-Flächen interpoliert werden, welche die Kurven miteinander verbinden. Durch die Manipulation der B-Spline Kurve mit Hilfe ihrer Kontrollpunkte lässt sich die entstandene Fläche so beeinflussen, dass sie die gewünschte Form erhält.

Dieser Vorgang wird in Abbildung 22 anhand verschiedener, aus der 3D Studio Max Oberfläche entnommener Ansichten veranschaulicht. Die beschriebenen Kurven (weiß dargestellt) zeigen die Grundlinien der Geometrie, welche an einigen Stellen mit NURBS-Flächen (grün dargestellt) verbunden sind.

---

[8] Non Uniform Rational B-splines.



a) oben *b) vorne*

c) links *d) Perspektive*

Abbildung 22: Grundstruktur der Gewölbedecke im Chor der Stiftskirche.

Anschließend kann die gezeichneten B-Spline Kurven (die Grundlinien der Gewölbekuppel) geklont und ihnen eine bestimmte Dicke zugeteilt werden. Die so entstandenen Bögen erhalten nun ebenfalls die Sandsteintextur. Dadurch kann der dem Original entsprechende Eindruck, dass die Deckenflächen von einem gemauerten Bogen gestützt sind, vermittelt werden. Sind alle Elemente der Decke fertig modelliert, werden die NURBS-Kurven und -Flächen in ein Poly-Modell konvertiert. Durch Parameter in der Oberflächenapproximation ist es möglich, Einstellungen an der Tesselierung vorzunehmen. Somit kann die Anzahl der Dreiecke, welche bei der Approximation der Flächen entstehen, auf ein gewünschtes Maß reduziert werden. Die in Abbildung 23 gezeigte Kuppel besteht aus ca. 11000 Polygonen und wurde mit drei unterschiedlichen Texturen belegt.

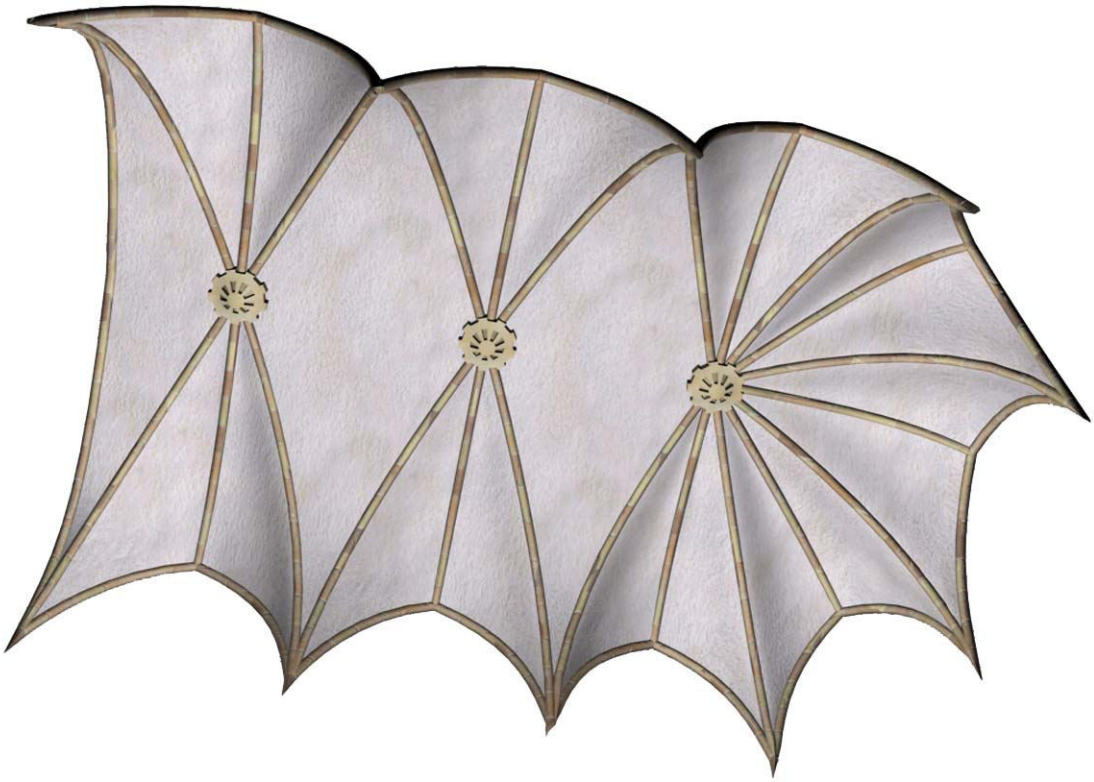


Abbildung 23: Kuppeldecke des Chors mit Textur.

5 Das architektonische Modell der Stiftskirche

Die in Kapitel 4 beschriebenen Methoden sind Beispiele für einige Möglichkeiten, welche 3D Studio Max zur Objekterstellung bietet. Das architektonische Modell der Stiftskirche wurde mit den an dieser Stelle beschriebenen Arbeitsweisen unter Beachtung der in Kapitel 3.2.1 erläuterten Kriterien der Modellierung erstellt. Hierbei wurden alle zum Gebäude gehörenden Räume der echten Stiftskirche wie zum Beispiel die Gruft unter dem Chor, die Räumlichkeiten und Treppenaufgänge der beiden Türme, die Galerie und ihre Wendeltreppen aus Glas und Stahl, die Fensterverzierung usw. so realitätsgetreu wie möglich nachempfunden und sind somit auch voll begehbar. Der Geometrieumfang des architektonischen Modells beträgt ca. 300 000 Polygone. Um den realistischen Eindruck zu vervollständigen, wurden die Geometrien mit mehr als 60 unterschiedlichen Materialien und Texturen versehen. Der Großteil dieser Texturen für die Kirchenfenster, Eingangstüren, Wände und Steine ist mit Hilfe der bei den Begehungen aufgenommenen Digitalbildern entstanden.

Der anschließende Abschnitt vermittelt den geschichtlichen Hintergrund der Stiftskirche. Mit den darauf folgenden Abbildungen soll ein Überblick über die Komplexität und Dimension des entstandenen 3D-Modells der Stiftskirche Stuttgart gegeben werden.

Geschichte der Stiftskirche Stuttgart:

In ihrer heutigen Gestalt ist die Stuttgarter Stiftskirche nicht nur das Ergebnis einer langen Entstehungszeit, sondern auch eines vereinfachten Wiederaufbaus nach schwersten Kriegszerstörungen.

Die Baugeschichte der evangelischen Pfarrkirche umspannt das gesamte hohe und späte Mittelalter. So reicht die ehemalige Stiftskirche in ihrer Gründungsgeschichte wohl bis in das frühe 11. Jahrhundert zurück. Von diesem in spätromanischer Zeit umgebauten Vorgängerbau ist lediglich das Untergeschoss eines in der zweiten Hälfte des 12. Jahrhunderts entstandenen Turmes erhalten. Im Zuge der um 1240 durch Graf Ulrich I. erfolgten Stiftung einer spätromanischen Basilika erfuhr auch dieser, ursprünglich als Chorturm fungierende Baukörper, umfassende Veränderungen.

Die 1321 erfolgte Überführung der gräflichen Grablage aus Beutelsbach und die damit verbundene Rangerhöhung der Kirche gab den Anlass für einen hochgotischen Chorneubau. Architekt des 1327 bis 1347 errichteten Chors war der auch am alten Schloss tätige

Meister Walther. Rund ein Jahrhundert später wurde auch das romanische Langhaus durch eine dreischiffige Halle der Spätgotik ersetzt. Als entwerfender Architekt des 1433 begonnenen Langhauses ist Hänslin Jörg überliefert, dessen Sohn Aberlin den Neubau 1495 vollendete. Noch im selben Jahr begann Meister Marx mit der Errichtung eines mächtigen Westturms, dessen Bau jedoch im Zuge der sich ankündigenden Reformation zum Erliegen kam. Die bereits gegen Ende des 15. Jahrhunderts erfolgte Aufstockung des südöstlichen Turmes fand erst 1578 mit dem Aufsetzen des Helms ihren Abschluss. Am 16. Mai 1534 fand in der Stiftskirche der erste protestantische Gottesdienst statt.

Die mittelalterliche und frühneuzeitliche Bausubstanz blieb über nahezu 400 Jahre erhalten, bis der schwere Luftangriff des Jahres 1944 der Stiftskirche schwerste Schäden zufügte. Einzig die nördliche Umfassungsmauer, Teile des Chors sowie die beiden Türme blieben erhalten. Der 1958 beendete Wiederaufbau zeigt eine überaus problematische Vereinfachung und Veränderung des ursprünglichen Bestandes. Eine erneute, knapp vierjährige Renovierung wurde im Juli 2003 abgeschlossen. Diese Arbeiten brachten Fundamente der romanischen Vorgängerbauten sowie mehrere Grabmale zum Vorschein [sch].

Abbildung 24: Nordseite der Stiftskirche.

Abbildung 25: Südseite der Stiftskirche.

Abbildung 26: Choranbau der Stiftskirche.

Abbildung 27: Hauptschiff der Stiftskirche.

5.1 Informationsbehaftete Modelle

Die Stiftskirche Stuttgart ist ein Bauwerk mit einer sehr interessanten und vielfältigen Vergangenheit. Da diesbezügliche Informationen dem Betrachter während der virtuellen Begehung bereitgestellt werden sollen, beschäftigt sich der folgende Teil dieser Arbeit mit der Aufbereitung dieser Inhalte und den dazugehörigen 3D-Modellen. Träger der Informationen sind die in der Kirche vorhandenen geschichtlichen Zeugen wie Grabmäler, Wandfresken, Standbilder oder Statuen. Um einen Besuch der virtuellen Stiftskirche noch interessanter zu gestalten, wurden einige dieser Objekte authentisch nachmodelliert und so aufbereitet, dass man sich direkt über sie informieren kann. Im Folgenden sind diese informationsbehafteten Modelle mit ihren zugehörigen Informationen beschrieben, wobei das Konzept zur Vermittlung der Informationen in Kapitel 6.3 ausführlich erklärt wird.

5.1.1 Die neue Orgel

Die Ende August 2004 eingeweihte Orgel ist das neue Herzstück der Stiftskirche. Schon im Jahre 1381, rund 200 Jahre früher als in anderen Stuttgarter Kirchen, wurde die erste Orgel in der Stiftskirche eingebaut. Weitere Orgeln folgten 1621 und 1668. Da sie sich zu dieser Zeit noch auf dem Lettner (Chorschranke) befanden, mussten sie entsprechend klein gestaltet werden. 1807 sollte die Stiftskirche jedoch auf Wunsch von König Friedrich I. eine wahre Prachtorgel erhalten. So ließ er die aus 64 Registern bestehende Orgel der Klosterkirche Zwiefalten ausbauen und mit der Hilfe von 26 Vierspännern nach Stuttgart transportieren. Der berühmte Orgelbauer Eberhard Friedrich Walcker setzte das Instrument schließlich auf die Westempore, wo sie einen beeindruckenden neugotischen Prospekt erhielt und auf 80 Register erweitert wurde. Leider wurde die legendäre Orgel bei einem Bombenangriff im Jahre 1944 vollständig zerstört. Nach dem Krieg baute die Firma Walcker eine neue Orgel mit 86 Registern. Als die notwendig gewordene Restaurierung dieser Orgel sich allerdings als nicht mehr lohnenswert herausstellte, begann die Planung der heutigen Orgel. Von der ehemaligen Orgel konnten lediglich die großen und äußerst wertvollen Holzpfeifen für das Pedal wieder verwendet werden. Für den kostspieligen Bau der neuen Orgel waren 5000 kg Orgelmetall und ein kleiner Mischwald notwendig. Elektronik wurde nur eingebaut, um Klänge speichern und schnell aufrufen zu können – im "Herz" der Orgel ist alles mechanisch. Ein großer Vorteil dieser Orgel ist die neue Akustik der Stiftskirche, durch die der Klang der Pfeifen rund und hell in den Raum getragen wird [sch].

Aus den vorangegangenen Erklärungen sollte deutlich geworden sein, dass dieses Modell ein wesentlicher Teil der Stiftskirche ist. Aus diesem Grund wurde es, wie in Abbildung 28 zu sehen ist, mit größter Sorgfalt modelliert.

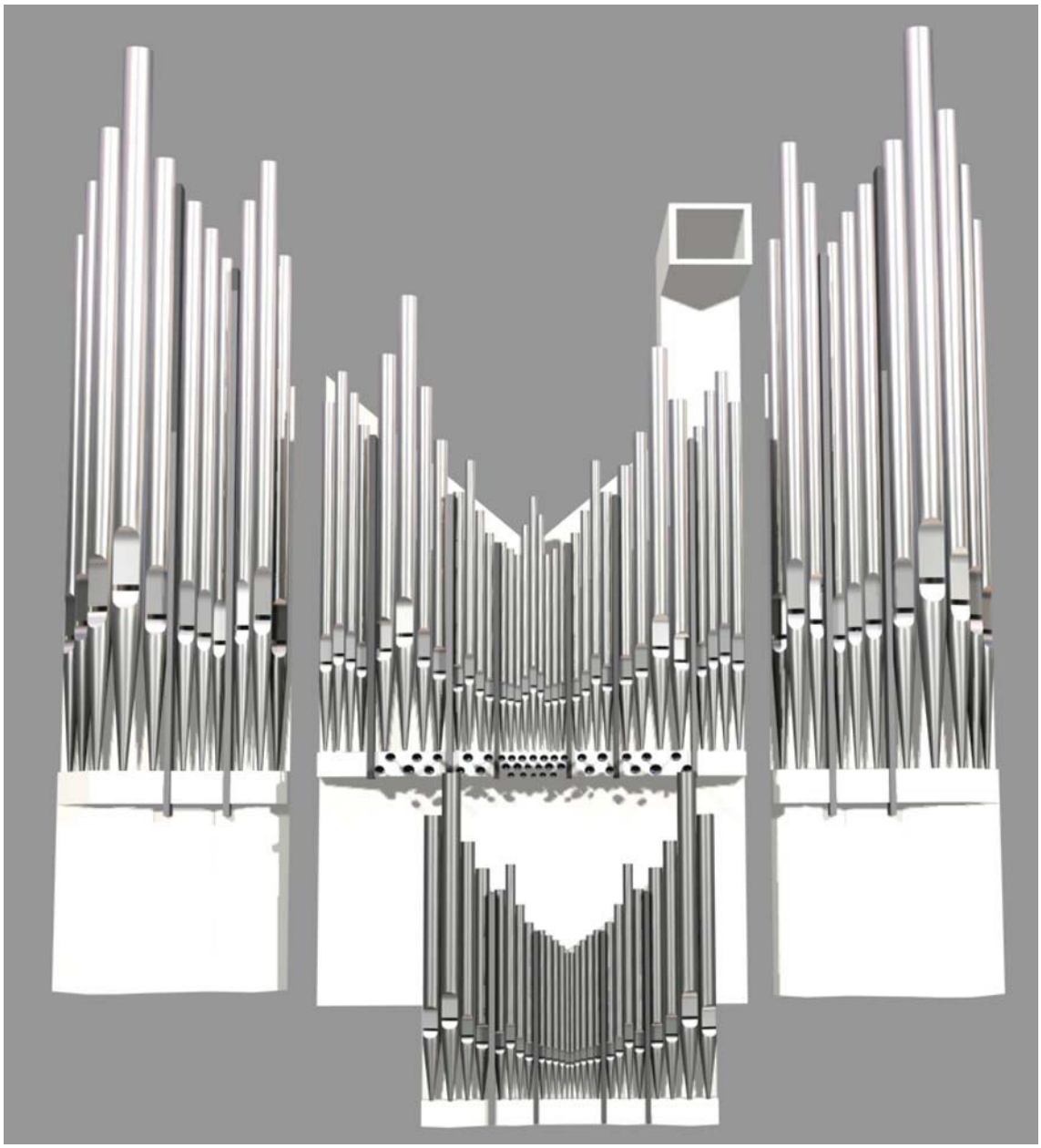


Abbildung 28: Das Modell der neuen Orgel.

5.1.2 Die Grafenstandbilder

1574 erteilte Herzog Ludwig Sem Schlör aus Schwäbisch Hall den Auftrag, die Grafenstandbilder an der Nordwand des Chors als „Ludwigs Ahnengalerie“ zu schaffen. Ursprünglich sollten nur die an der Chorwand aufgestellten Grabmale restauriert werden, doch wurde schließlich dieses den Chor bis heute prägende Renaissancekunstwerk gestaltet. Die Standbilder stellen württembergische Grafen dar, die jeweils in ihrer historisch

korrekten Rüstung abgebildet sind. Ursprünglich waren die Standbilder aus Stein farbig gefasst, Waffen und Amts-Attribute sind aus Metall. Name, Geburts- und Todesjahr des Ritters sind jeweils auf dem Gesims oberhalb der Nischen in eine schwarze Marmortafel eingemeißelt und darüber befindet sich das dem Grafen zugehörige Wappen. Zwischen den Standbildern wurden schildhaltende Putten aufgestellt. Die Grafen stehen auf Löwen (Heldensymbol) vor flachen Bogennischen. Dargestellt sind nur diejenigen Grafen, die in direkter Nachfolge stehen und in der Stiftskirche auch begraben wurden [sch].

Abbildung 29: Foto des Grafenstandbildes.

Das Foto des Grafenstandbildes aus „Die Stiftskirche in Stuttgart“ [sor] verdeutlicht die Komplexität der Skulpturen (s. Abbildung 29). Um diese auf ein Modell zu übertragen, war es zunächst notwendig, ein digitales Photo von den jeweiligen Grafen auf dem Standbild zu

machen. Anschließend konnten diese mit Photoshop farblich aufeinander angepasst und aneinandergehängt werden. Die dabei entstandene Textur besitzt einem Umfang von 1024 x 3072 Pixel und wird, um die transparenten Stellen darstellen zu können, im SGI rgb-Format mit Alphakanal gespeichert. Da die Figuren auf der Textur sehr plastisch erscheinen, konnte das 3D-Modell des Standbildes sehr reduziert modelliert werden.

Abbildung 30: Modell des Grafenstandbildes.

5.1.3 Die Grabplatte des Stiftspropstes Ludwig Vergenhans

Ludwig Nauclerus (alias Vergenhans) stammt aus einer Familie ritterlichen Standes. Jedoch wird vermutet, dass erst seinem Vater, Johannes Ferge (Vergenhans), der Aufstieg in den Ritterstand gelang. Der ältere Bruder Ludwigs, Johannes Nauclerus, war der Gründungsrektor und spätere Kanzler der Tübinger Universität. Informationen zur schulischen Bildung und zum Studium von Ludwig Nauclerus vor 1467 sind nicht überliefert. Die vielen Parallelen im Lebenslauf der beiden Brüder legen jedoch nahe, dass Ludwig Nauclerus eine ähnliche akademische Ausbildung wie sein Bruder durchlief. Als der württembergische Graf Ulrich V. ihm 1471 für seine treuen Dienste die Pfründe der Pfarrei Waiblingen versprach, besaß Ludwig Nauclerus bereits die Magisterwürde. Verdient gemacht hatte sich Nauclerus als Erzieher der beiden Söhne des Grafen, Heinrich und Eberhard der Jüngere. 1467 begab sich Nauclerus als Begleiter Heinrichs auf eine zweijährige Auslandsreise, welche ihn unter anderem nach Pavia und Ferrara führte. Dort wurde er 1469 zum Doctor utriusque iuris promoviert.

Abbildung 31: Die Grabplatte Ludwig Vergenhans.

Wann genau Ludwig Nauclerus in den geistlichen Stand eintrat ist unbekannt. 1468 erhielt er, mit Erlaubnis zur Abwesenheit, das Amt eines Kaplans an der Waiblinger Pfarrkirche, 1469 die Pfarrei in Calw und von 1472 bis 1488, als Nachfolger seines Bruders Johannes, die Pfarrei in Kirchheim. Die Propstei der Stuttgarter Stiftskirche, an der sein Bruder nur wenige Jahre zuvor dasselbe Amt innegehabt hatte, versah er von 1483 bis zu seinem Tod im Jahre 1512. Daneben hatte Vergenhans zahlreiche weitere Pfründe inne, unter anderem die Kanonikate der Domkapitel Konstanz und Worms. Sein Grabmal befindet sich in der Stuttgarter Stiftskirche [sch].

5.1.4 Das Epitaph des württembergischen Reformators Johannes Brenz

Das Ölbild des Reformators wurde 1584, 14 Jahre nach seinem Tod, von Jonathan Sautter aus Ulm gemalt. Dessen Umrahmung wurde 1944 zerstört und durch eine Nachbildung ersetzt. Rechts ist der Tod mit gespannter Armbrust abgebildet, links der auferstandene Christus mit der Kreuzesfahne als Sieger über den Tod, eine Darstellung, die an die Grundlage des christlichen Glaubens erinnert.

Abbildung 32: Epitaph des Egon Brenz.

Oberhalb steht übersetzt: „Das Wort des Herrn ist eine Leuchte meinen Füßen und ein Pfad des Lebens“ (nach Psalm 119, 105). Unterhalb steht: „Durch Wort, Schrift, Frömmigkeit, Glauben, Lauterkeit bewährt, ist Johannes Brenz von solchem Antlitz gewesen“. Der Wortlaut der Grabschrift selbst lautet: „Gott dem Besten, Höchsten geweiht (Abkür-

zung D.O.M.S. = Deo optimo maximo sacrum). Johannes Brenz von schwäbischem Stamme, beheimatet in Weil der Stadt, hochberühmter Theologe, Probst in Stuttgart, erlauchtester württembergischer Herzoge Rat, war einer der ersten Prediger der wiedergereinigten Kirche. Die Schriften der Propheten und Apostel hat er in Schulen, bei heiligen Versammlungen, auf Tagungen des römischen Reiches sowie in seinem nächtlichen Studium erläutert und verfochten. Um des Bekenntnisses willen hat er die Verbannung standhaft ertragen und der Kirche und dem gemeinsamen Vaterland mit Ratschlägen beigestanden. Durch die Unbescholtenheit seines Lebens hat er sein Amt geziert und als er in seinem Lauf fünfzig und mehr Jahre zum großen Nutzen der Kirche gearbeitet hatte, ist er sanft entschlafen und wurde an dieser Stätte unter größter Trauer aller Frommen im Jahr 1570 im Monat September am 11. Tage bestattet, nachdem er 71 Jahre, 2 Monate und 17 Tage gelebt hatte“ [sch].

5.1.5 Die Gruft der Stiftskirche

Unter dem Chor und der Sakristei befinden sich zwei räumlich getrennte Grabkammern, in welchen die württembergischen Grafen seit Graf Eberhard bestattet wurden. Insgesamt befinden sich hier 66 Särge sowie weitere 40 Tote in einem Sammelbehältnis. In die ältere Grabkammer unter der Sakristei wurden 1321 die Gebeine der württembergischen Ahnen aus Beutelsbach überführt. Nach dem Tod Herzog Friedrichs im Jahre 1608 wurde unter dem Chor innerhalb von 17 Tagen ein neues Grabgewölbe errichtet und die Gebeine der bisher bestatteten Toten wurden in einem besonderen Behältnis geborgen. 1683 fand eine Erweiterung der Gruft unter der Sakristei statt. Zusätzlich wurde eine Verbindung mit der Chorgruft hergestellt [sch].

Die Geometrien der unterschiedlichen Särge sind aus dem Internet geladene 3D Studio Max Geometrien (frei verfügbar unter http://www.3dcafe.com/asp/horror.as).

Abbildung 33: Die Gruft der Stiftskirche.

5.2 Interaktive Modelle

Nachdem nun die informationsbehafteten Modelle innerhalb der Stiftskirche erläutert wurden, befasst sich der folgende Teil der Arbeit mit Objekten, welche als interagierbare Modelle ausgelegt sind. Zunächst wird jedoch nur beschrieben, in welcher Form sich die Interaktion äußert, während die Art und die einzelnen Funktionen der Implementierung in Kapitel 7 ausführlich erklärt werden.

5.2.1 Bibel auf dem Altar

Da der Altar der Mittelpunkt jeder Kirche ist, wurde ihm auch hier eine besondere Eigenschaft zuerkannt. Die Texturen des Altars wurden auf eine besondere Weise erzeugt und weisen einen Schattenwurf der darauf stehenden Kerzen und der Bibel auf. Die 3D Studio Max Funktion 'render to texture' macht es möglich, Schatten auf einer Fläche in deren Textur einzurechnen und diese anschließend als eigene Textur abzuspeichern (shadow map). Damit diese Informationen auch in das VRML Format exportiert werden können, werden die entstandenen Texturen wiederum in einem Material verwendet und anschließend auf die einzelnen Flächen gemappt. Allerdings ist es sehr zeitaufwendig, diese Textu-

ren zu generieren. Zudem ist die Anzahl der erzeugten Texturen abhängig von den Flächen, aus denen sich das Modell zusammensetzt, da jede Fläche eine eigene Textur zugewiesen bekommt. Am Altar sollte diese Funktion jedoch beispielhaft aufgezeigt werden.

Auf dem Altar befindet sich eine sehr schön illustrierte Bibel aus dem Mittelalter. Diese ist so ausgelegt, dass sie im virtuellen Raum in die Hand genommen, frei im Raum bewegt und dabei näher betrachtet werden kann.

Abbildung 34: Die Bibel auf dem Altar.

5.2.2 Das Taufbecken

Das Taufbecken eignet sich sehr gut, um Wassereffekte zu visualisieren. Die Idee dabei ist, dass sich während der Simulation eine Welle über die Wasseroberfläche ausbreitet, wenn der Anwender vor dem Becken steht und mit der Hand die Wasserfläche berührt. Die Wasserfläche hat einen sehr hohen Geometriebedarf, da die animierte Wasserfläche bei zu großen Dreiecken eckig und unecht erscheinen würde. Aus diesem Grund ist die Geometrie für den Steinsockel und das Becken sehr einfach gehalten.

Abbildung 35: Das Taufbecken mit Wasseroberfläche.

5.2.3 Die neue Orgel

Da die Orgel bei einer Besichtigung in der CAVE™ nicht nur visuell, sondern auch auditiv ein Erlebnis sein soll, wurde es ermöglicht, durch Berühren der Orgel ein musikalisches Orgelstück von Bach wiedergeben zu lassen.

5.2.4 Sarg mit Skelett

In einem der Särge in der Gruft ist das Modell eines Skeletts (frei verfügbar aus dem Internet unter http://www.3dcafe.com/asp/anatomy.asp) vorhanden. Um die Idee der Interaktion hierbei umzusetzen, wurde ein Trick der Computergrafik angewendet. Dieser macht es möglich, die Wand des Sargs mit einer „X-Ray Taschenlampe“ zu durchleuchten und so das Skelett darin begutachten zu können. Die Funktionsweise der X-Ray Taschenlampe wird in Kapitel 7 ausführlich erklärt. Das Bild in Abbildung 36 soll den Effekt der Durchleuchtung annähernd darstellen.

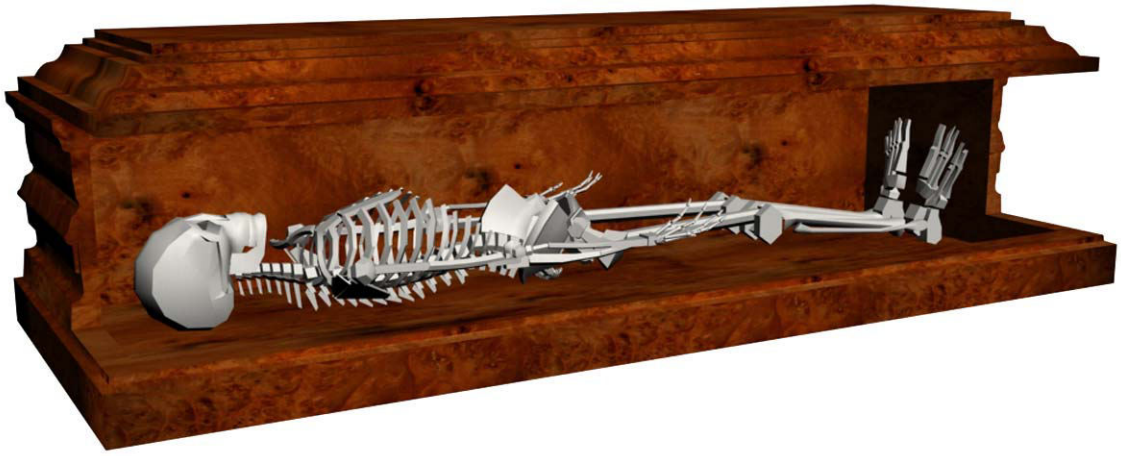


Abbildung 36: Sarg mit Skelett.

5.3 Umgebungsterrain der Stiftskirche

Das Gebäude der Stiftskirche soll nicht nur im Inneren begehbar, sondern auch von außen zu begutachten sein. Um dies zu gewährleisten wird eine Umgebung, welche mit dem Stadtgebiet in der unmittelbaren Nähe der Kirche übereinstimmt, benötigt. Im Fraunhofer Institut gibt es ein Modell aus Vermessungsdaten vom Innenstadtbereich Stuttgarts. Da diese Geometrie jedoch von vielen Löchern durchzogen und fragmentiert ist, konnte sie nur als maßstabsgetreue Vorlage dienen. Für die Bearbeitung wurde sie aus dem Performer pfb Format nach VRML konvertiert und anschließend in 3D Studio Max importiert. Nun war es möglich, mit einfachen Primitiven die nähere Umgebung mit dem Schillerplatz, dem alten Schloss, der Markthalle, der Fußgängerzone und den umgebenden Häusern nachzumodellieren. Um den Betrachter aber nicht zu sehr von dem Bauwerk der Stiftskirche abzulenken, wurden keine Texturen an die Hauswände und Dächer angebracht, sondern lediglich dezente Farbtöne auf Wände und Dächer vergeben. Nur zwei Texturen wurden verwendet, um den Eindruck wieder realistischer zu gestalten. So wurde die Himmelskuppel mit einer Textur aus Wolken und blauem Himmel versehen und der Boden weist eine Textur aus Pflastersteinen auf. Abbildung 37 zeigt die beschriebene Umgebung der Stiftskirche.

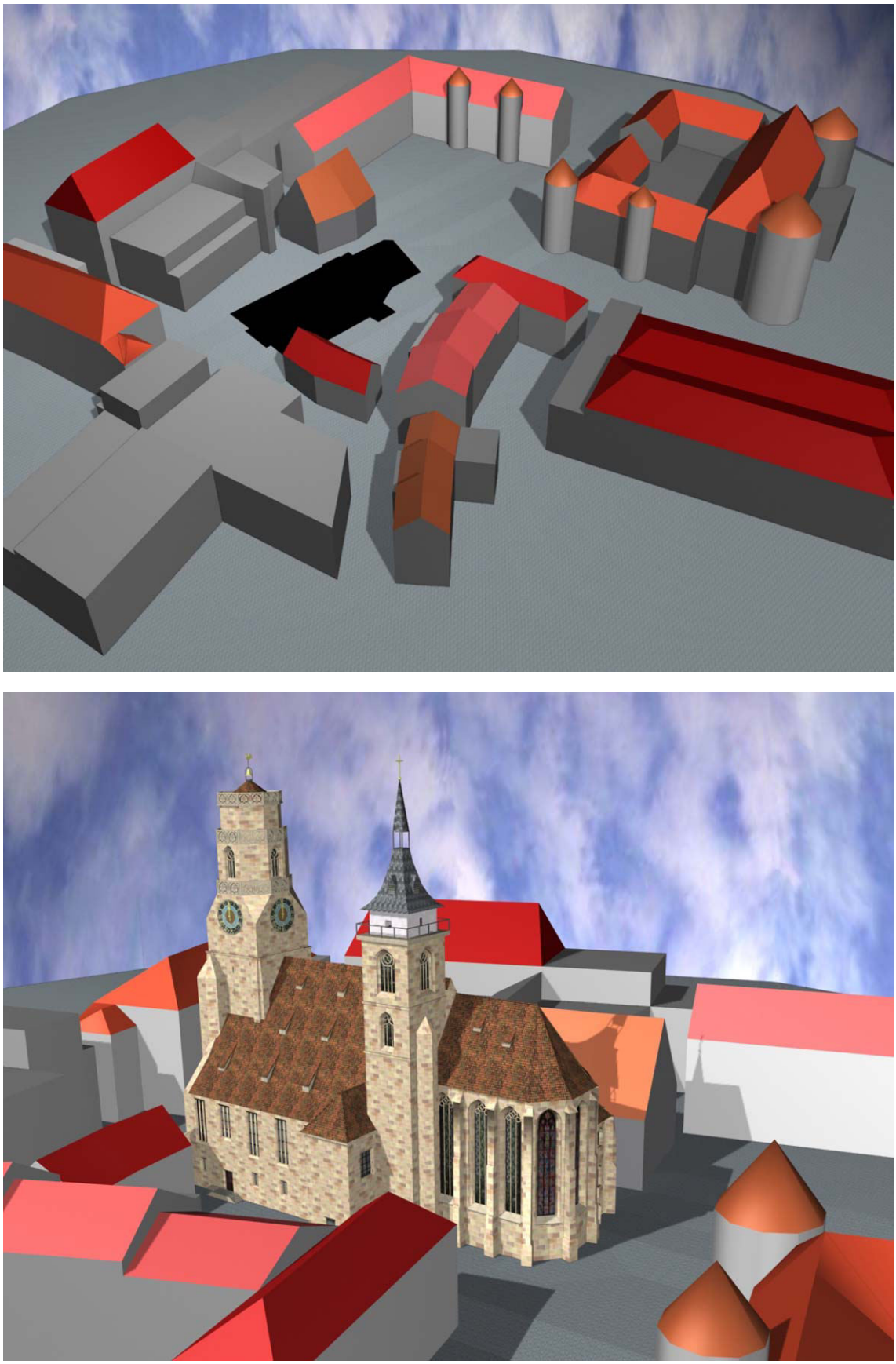


Abbildung 37: Umgebung der Stiftskirche.

6 Konzeption der VR-Applikation

In virtuellen Umgebungen stellen die Navigation im Raum sowie das Selektieren und Manipulieren von Objekten grundlegende Anforderungen an die Anwendung. Des Weiteren besitzt diese Arbeit den Anspruch, dem Anwender zusätzliche Informationen, welche nicht als 3D-Geometrie dargestellt werden können, zu Verfügung zu stellen. Die Interaktionen sind so ausgelegt, dass der Anwender alle Möglichkeiten der Interaktion mit nur einem Eingabegerät steuern kann. Dies funktioniert über drei unterschiedliche Modi: dem Navigationsmodus, dem Interaktionsmodus und dem Informationsmodus, die vom Anwender ausgewählt werden. Jeder Modus besitzt eigene Konzepte, welche im Folgenden beschrieben werden.

6.1 Das Navigationskonzept

Unter Navigation versteht man eine Interaktionstechnik, die es dem Anwender ermöglicht, sich durch die dreidimensionale Szene zu bewegen. Durch die Änderung von Position und Orientierung der virtuellen Kamera ist eine Bewegung des Sichtpunktes möglich. Über Tracking kann der Anwender minimale Distanzen in der virtuellen Welt mit korrespondierenden Bewegungen in der realen Welt (z.B. innerhalb der CAVE™) bewältigen. Bewegungen über weitere Distanzen benötigen eine andere Technik der Navigation, zum Beispiel die Steuerung der Bewegung über Eingabegeräte. Dabei existieren unterschiedliche Bewegungsmöglichkeiten [stu96].

- **Relative Bewegung:** Die Änderung des Standpunktes durch den Anwender mit wenigen Schritten in der nichtsynthetischen Welt.
- **Absolute Bewegung:** Durch Interaktion des Anwenders mit einem Eingabegerät wird eine kontinuierliche Positionsänderung in der virtuellen Umgebung berechnet.
- **Teleportieren:** Direkter Transport (ohne fließende Bewegung) zur gewünschten Position in der virtuellen Umgebung. Die gewünschte Position kann zum Beispiel auf einer Karte angegeben werden.

Teleportieren ist eine für den Betrachter sehr unrealistische Fortbewegungsmethode, da sie dieser ohne zeitlichen Versatz zum neuen Standort befördert. Aufgrund des Umfangs der zurückzulegenden Distanzen in der virtuellen Stiftskirche wird in dieser Arbeit für die Navigation nicht nur die relative Bewegung, welche automatisch von Lightning auf das

Kameramodell übertragen wird, verwendet, sondern zusätzlich eine Methode zur absoluten Bewegung entwickelt. Dabei ist zu beachten, dass die Handhabung der Navigationswerkzeuge so intuitiv wie möglich gehalten werden soll, damit der Anwender seine Aufmerksamkeit der eigentlichen Aufgabe widmen kann und nicht zu stark mit der Navigation beschäftigt ist [bow04]. Für die Navigation durch die virtuelle Stiftskirche werden die folgenden zwei Arten der Fortbewegung (Bewegungs-Metaphern[9] [ket93]) benötigt.

6.1.1 Flying

Der Anwender hat die Möglichkeit, sich fliegend durch die Szene zu bewegen, indem er einen Knopf am Input Device drückt und dieses in die gewünschte Richtung bewegt. Mit der Umrechnung der Eingaben in eine Bewegung verändert der Standpunkt (Viewpoint) seine Position. Je stärker der Anwender die Position des Devices verändert, desto schneller wird die Bewegung durch den Raum. So kann der Anwender sich in allen Achsrichtungen bewegen und seine gewünschte Position ansteuern. Durch die Drehung des Gerätes wird dessen Orientierung ebenfalls auf den Standpunkt übertragen. Dadurch ist es dem Betrachter möglich, seine Ansicht im virtuellen Raum um alle Achsen zu drehen.

Mit den 6 Freiheitsgraden (degrees of freedom oder DOF) sind allerdings viele VR-Anwender überfordert. Daher werden die Freiheitsgrade um eine Achse der Orientierung beschränkt, wodurch eine einfachere Navigation gewährleistet wird. In diesem Beispiel bietet es sich an, auf den Roll-Freiheitsgrad zu verzichten, sodass der Betrachter sich nach rechts und links (Yaw) sowie nach oben und unten (Pitch), jedoch nicht mehr um die Roll-Achse drehen kann. Somit wird ein leichteres Navigieren ermöglicht.

Die Bewegung durch die Szene mittels der Fly-Metapher erlaubt dem Betrachter eine absolut freie Erkundung der Stiftskirche aus allen Perspektiven, auch der im Original unzugänglichen Stellen. Hierbei werden sein Blickwinkel und seine Position nicht speziell auf die informationsbehafteten Objekte gelenkt, wie dies bei der später erläuterten geführten Tour der Fall ist.

---

[9] Der Begriff Metapher stellt eine Redewendung dar, in der statt der eigentlichen Beziehung eine übertragene gebraucht wird, um einen Sachverhalt zu beschreiben wie z.B. das Übertragen von Vorgängen aus der Realität nach VR.

6.1.2 Walking

Wie bei der Fly-Metapher steuert der Anwender die Bewegung auch hier mit seiner Eingabe am Interaktionsgerät durch die Szene. Jedoch gibt es einige Besonderheiten bei dieser Methode. Der Freiheitsgrad der Orientierung wird hier um einen weiteren Grad (Pitch) eingeschränkt, sodass der Anwender sich in der Szene nur noch nach rechts und links drehen kann. Dadurch wird ihm die Kontrolle der Fortbewegung um ein Vielfaches erleichtert. Diese Reduktion der Bewegungsfreiheit wird vom Benutzer jedoch nicht als störend empfunden, da er mit seiner relativen Bewegung (z.B. Kopfdrehung) immer noch in alle Richtungen blicken kann. Des Weiteren bewegt sich der Betrachter stets mit dem definierten Abstand seiner Körpergröße über den Boden, wodurch, wie der Name schon sagt, ein gehender Bewegungseindruck entsteht. Eine besondere Anforderung ist, dass der Anwender das Gebäude nicht nur auf der Grundebene, sondern auch über alle Treppen, Schrägen und Flächen erkunden kann und hierbei immer den Eindruck behält, sich gehend durch dieses hindurch zu bewegen. Auch die Einschränkung, dass der Anwender keinen direkten Einfluss auf seine Höhe besitzt, wird von diesem nicht als störend, sondern als hilfreich bei der Orientierung empfunden, da er sehr viel weniger genau mit der Navigation umgehen muss. Diese Navigationsmethode bietet sich besonders in engen Räumen an oder wenn der Anwender Aufgänge oder bestimmte Objekte präzise ansteuern muss.

6.1.3 Geführte Tour

Um dem Anwender die Wahrnehmung aller wichtigen und interessanten Objekte zu gewährleisten, besteht die Möglichkeit einer geführten Tour durch die Stiftskirche. So kann der Betrachter sich 'per Knopfdruck' automatisch zum nächsten informationsbehafteten Objekt transportieren lassen, ohne selbst dorthin navigieren zu müssen.

Der Transport soll jedoch nicht mit einem Teleport (sehr plötzlicher Positionswechsel) erfolgen, da der Anwender damit seine räumliche Orientierung verlieren kann, sondern in einer kontinuierlichen Bewegung vom Start- zum Endpunkt ausgeführt werden [bow04]. Die geführte Tour kann an jedem beliebigen informationsbehafteten Objekt begonnen, weitergeführt, wieder aufgenommen oder abgebrochen werden, wodurch der Anwender selbst Herr der Navigation und nicht Sklave der Anwendung ist. Daher besitzt die geführte Tour keinen definierten Anfangs- oder Endpunkt, sondern ist in einer Schleife über alle informativen Objekte angeordnet.

6.1.4 World In Miniature

Das Gebäude der Stiftskirche ist nicht einfach zu überschauen, da es mehrere Aufgänge zu Emporen, Ein- und Ausgangstüren und einige verwinkelte Stellen gibt. Hinzu kommt, dass der Betrachter durch die Navigationstechnik verwirrt werden kann, da er im Flugmodus beispielsweise Positionen und Orientierungen einnimmt, die ihm in der Wirklichkeit aufgrund der Schwerkraft verwehrt bleiben. Um eine Verwirrung des Betrachters zu vermeiden, ist deshalb unbedingt ein Orientierungspunkt für diesen zu errichten.

Virtual Reality bietet zur Vermittlung des Raumverständnisses die besten Voraussetzungen, da es hiermit möglich ist, ein kleines vereinfachtes Modell der Stiftskirche mit transparenten Wänden als Orientierungshilfe zu visualisieren. Das Modell, die World In Miniature WIM, ist eine Art dreidimensionale Karte des Gebäudes und bewegt sich mit dem Betrachter durch den Raum. Die Orientierung der World In Miniature korrespondiert mit dem Sichtfeld des Betrachters in der Stiftskirche. Der Betrachter kann so mit einem Blick auf das kleine Abbild der Stiftskirche erkennen, in welche Richtung er sich gerade orientiert, was ihm vor allem im Fly-Modus ein erleichtertes Zurechtfinden garantiert. Des Weiteren wird der Standpunkt des Betrachters durch einen kleinen roten Punkt innerhalb der Miniatur-Stiftskirche markiert. Dieser erleichtert dem Betrachter seine Wegeplanung. Zusätzlich wurden die Positionen der interaktiven und informationsbehafteten Modelle als grüne Kugeln in die WIM integriert, damit sie im Modell leichter zu finden sind. In der folgenden Abbildung ist das Modell der World In Miniature der Stiftskirche mit den jeweiligen Repräsentanten zu sehen.

Die WIM ist so ausgelegt, dass sie durch weitere Optionen erweiterbar ist. Beispielsweise ließe sich dadurch eine Interaktion mit der World In Miniature realisieren. Da der Anwender einen Repräsentanten in der Szene besitzt, könnte er damit 'sich selbst' greifen und an eine neue Position in der Umgebung bewegen [sto95].

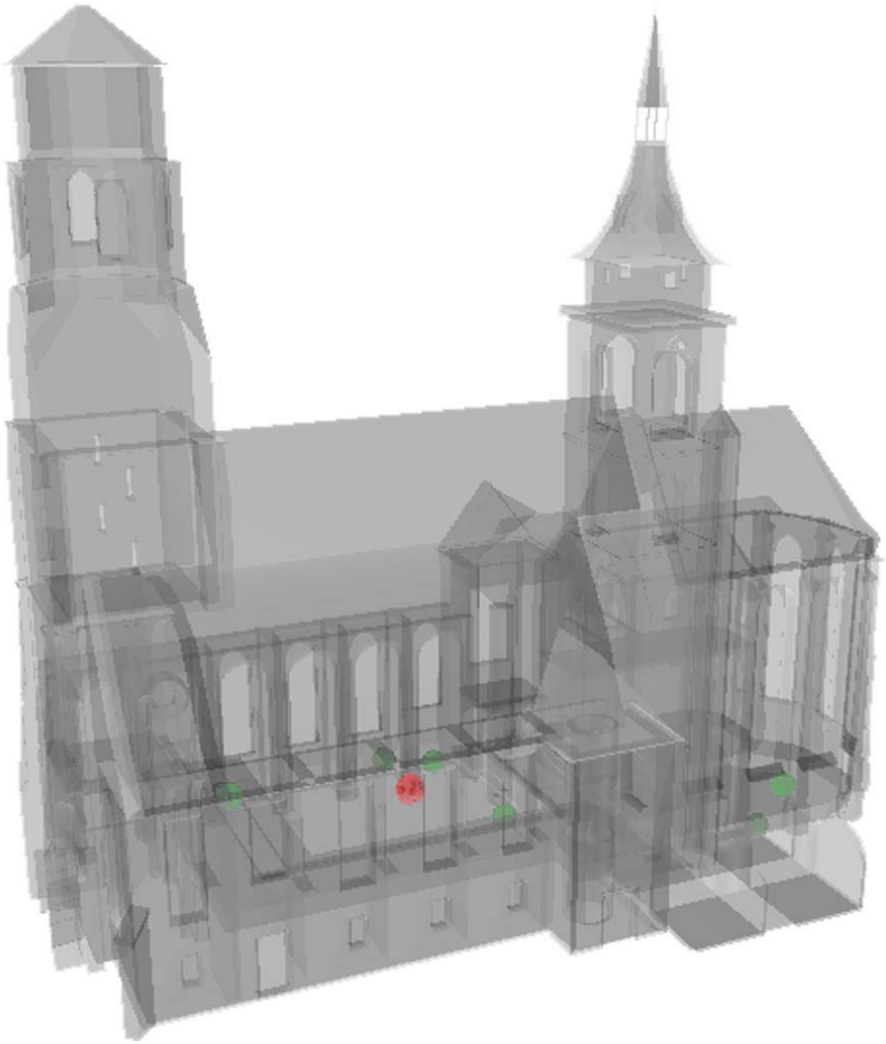


Abbildung 38: World In Miniature der Stiftskirche.

6.2 Das Interaktionskonzept

Bei der Interaktion mit den in Kapitel 5.2 beschriebenen Modellen stehen mehrere Operationen im Mittelpunkt. Diese reichen von der Translation und Rotation einzelner Objekte zu sehr speziellen Operationen wie Shader- und Audioansteuerung. Jedes interaktive Objekt in der Stiftskirche besitzt eine eigene Möglichkeit der Interaktion. Um den Anwender bei seiner Erkundung zu unterstützen, sind die unterschiedlichen Interaktionen an die jeweiligen Modelle gebunden. Somit muss er die gewünschte Art der Manipulation nicht aus einem Menu auswählen. Vielmehr steht ihm in dem Augenblick, in dem er ein interaktives Objekt erreicht, das geeignete Interaktionswerkzeug zu Verfügung, sodass er dieses sehr intuitiv einsetzen kann.

6.2.1 Greifen und Selektieren

Interaktionstechniken zur Manipulation von 3D-Objekten in virtuellen Umgebungen sollten Funktionen bereithalten, die es ermöglichen grundlegende Aufgaben auszuführen, wie das Selektieren, das Positionieren und das Rotieren eines Objekts im Raum. Die Art und Weise der direkten Manipulation mit der Hand hat sich nicht nur in der realen, sondern auch in der virtuellen Welt als eine prinzipielle Interaktionstechnik herausgestellt. Die Umsetzung der Techniken zur Selektion und Manipulation haben großen Einfluss auf die Qualität des gesamten Userinterfaces [bow01].

Die klassische Methode, welche auch in dieser Arbeit umgesetzt wurde, ist die Bereithaltung einer virtuellen Hand, die in ihrer Bewegung und Orientierung mit der Hand des Benutzers korrespondiert. Die Selektion erfolgt, indem der Anwender das Objekt berührt und einen Button am Eingabegerät drückt. Das Objekt 'hängt' somit an der Hand des Benutzers und kann auf diese Weise in eine beliebige Richtung und Position gebracht werden.

Ist die aktuelle Position des Anwenders nun vor dem Altar, so steht ihm eine Greifoption auf das sich darauf befindende Buch zu Verfügung. Ist dieses mit der virtuellen Hand ausgewählt, lässt es sich sehr intuitiv bewegen und an einer anderen Stelle ablegen, da diese Metapher eine Interaktion aus der realen Welt simuliert.

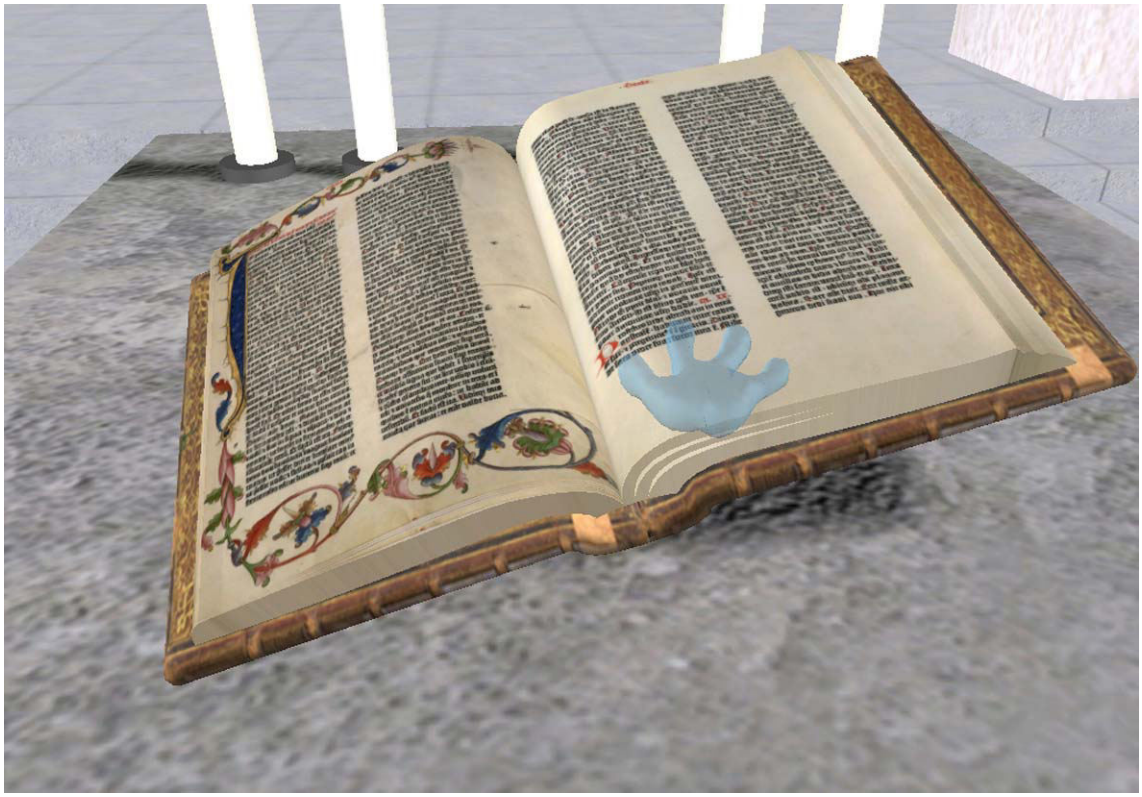


Abbildung 39: Interaktion mit dem Buch auf dem Altar.

6.2.2 Aktion auslösen durch Berührung

Befindet sich der Anwender am Taufbecken, so wird die virtuelle Hand durch einen Wassertropfen ersetzt, mit welchem er die Wasseroberfläche im Taufbecken berühren kann. Allerdings wird hier keine Manipulation des Objekts in Form von Änderungen an der Position bewirkt, sondern ein Signal an ein spezielles (dynamisches) Geometrieobjekt gesendet, welches eine animierte Welle über die Wasserfläche gleiten lässt.

Auch die Interaktion mit der Orgel funktioniert in einer ähnlichen Weise. Berührt der Anwender die Orgel mit dem an der Stelle der virtuellen Hand angezeigten Notenschlüssel, so wird das Musikstück einer Orgelsonate von Bach abgespielt.

6.2.3 Durchleuchten der Sargwand

Um den Sarg zu durchleuchten, wird eine Schnittebene (Cutting Plane) verwendet, die es möglich macht, ein ausgewähltes Objekt zu zerschneiden. Ist die Position des Betrachters in der Nähe des Sarges, kann dieser eine X-Ray Taschenlampe zur Interaktion auswählen, welche ihm als Geometrie an seine Hand projiziert wird. In einem definierten Abstand zur X-Ray Taschenlampe befindet sich eine Cutting Plane, die nur auf eine bestimmte Geometrie, den Sarg, wirkt. Bewegt der Betrachter nun die Cutting Plane über den Sarg, so werden alle Polygone, die sich vor der Plane befinden, abgeschnitten. Anschließend wird die entstandene Schnittfläche geschlossen und mit der Textur des Sarges belegt. Der Betrachter hat damit die Möglichkeit, den Sarg von allen Seiten zu durchleuchten und das sich darin befindende Skelett zu begutachten.

6.3 Das Informationskonzept

Mit der Vermittlung von Metainformationen zu einzelnen visuellen Objekten hat der Anwender über die rein geometrische Visualisierung hinaus die Möglichkeit, sich mit der virtuellen Welt auseinanderzusetzen. Dazu muss die Information jedoch so aufbereitet werden, dass der Anwender sie in dem VE verwenden kann, ohne sich durch unkomfortable Menüs und Steuerelemente hangeln zu müssen. Eine geeignete Methode der Informationsvermittlung ist deren Transport über Ton. Da Sprachtools, die einen Text mittels einer computergenerierten Stimme wiedergeben, immer sehr künstlich klingen und eher als störend wahrgenommen werden, ist diese Weise der Informationsübertragung hier nicht geeignet. Um einen angenehmen Eindruck zu vermitteln, sollten somit alle Texte von einem professionellen Sprecher vorgetragen und dabei vertont werden. Da die Produktion einer solchen Vertonung jedoch sehr aufwendig ist, kommt sie in dieser Arbeit nicht zum

Einsatz. Stattdessen wird die Informationsvermittlung über lesbare Texte verwendet. Trotzdem bleibt die zuvor beschriebene Methode der Vertonung weiterhin als Option zur Erweiterung bestehen.

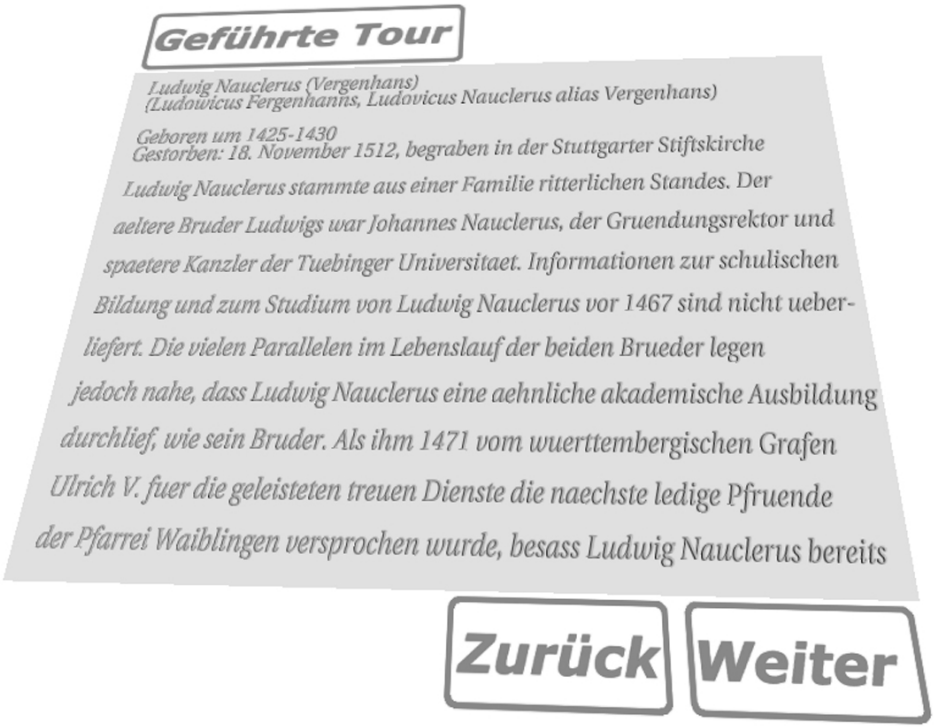


Abbildung 40: Das Informationsmenü.

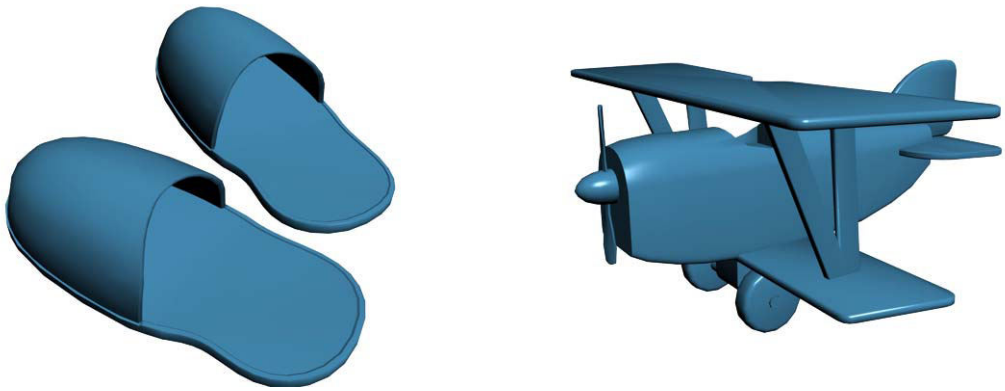
Der Informationstransfer anhand von Texten ist allerdings auch mit einigen Problemen verbunden. Das Lesen ist in einer virtuellen Umgebung schwerer und ermüdender als in der Realität. Deshalb wurde eine besondere Geometrie entworfen (ein Informationsmenü), auf welcher der Text angezeigt wird. Bei der Anwendung solcher 3D-Widgets ist besonders darauf zu achten, dass der Geometrieumfang sehr gering bleibt, das Widget leicht als Interaktionselement zu erkennen ist und sich von den anderen Objekten in der Szene unterscheiden lässt [lei97]. Das Informationsmenü folgt dem Benutzer durch das VE, ist jedoch nur an Objekten mit Informationsgehalt sichtbar und erreichbar. Zusätzlich kann der Anwender es zum Lesen frei an eine gewünschte Position bewegen. Da die Lesbarkeit des Textes äußerst wichtig ist, wurde der jeweils angezeigte Text mit einem großen Zeilenabstand versehen und auf eine bestimmte Zeilenanzahl begrenzt. Um mehr Informationen an ein Objekt binden zu können, hält das Informationsmenü die Möglichkeit bereit, mittels Schaltflächen über mehrere Seiten zu blättern. Das Informationsmenü dient der direkten Interaktion zwischen Benutzer und Inhalt. Es gleicht sozusagen einem Reiseführer, in welchem der Benutzer auf Wunsch Informationen zu einem bestimmten Objekt erhält.

Zusätzlich ist hier die Auswahlmöglichkeit der geführten Tour in Form einer Schaltfläche untergebracht, da der Anwender so auf Wunsch direkt vom aktuellen Objekt zum nächsten interessanten Objekt befördert werden kann.

Die notwendigen Inhalte liegen in einer Datenbank und werden bei Bedarf von dort gelesen und im Informationsmenü angezeigt. Der Vorteil einer Ablage der Inhalte in eine Datenbank ist die schnelle und unkomplizierte Möglichkeit der Erweiterung der Inhalte, falls weitere informationsbehaftete Modelle zu der Szene hinzukommen.

6.4 Piktogramme zur Statusvisualisierung

Damit der Betrachter während der Simulation immer weiß, welche Navigations- oder Interaktionsmöglichkeiten er ausführen kann, wird ihm sein aktueller Status in Form eines Piktogramms[10] visualisiert. Als wichtigstes gemeinsames Merkmal aller verwendeten Piktogramme ergibt sich deren typische Darstellungsweise. Sie ist relativ schnörkellos und abstrakt, losgelöst von allen graphischen Ausschmückungen. Ein ideales Piktogramm sollte demnach so beschaffen sein, dass es das Wesentliche repräsentativ für eine Klasse von Gegenständen oder Handlungen darstellt (vgl. [sta87]). Die in dieser Applikation selbst entwickelten Piktogramme zur Veranschaulichung der Interaktionsmöglichkeiten sind in den folgenden Abbildungen zu sehen.



a) *b)*

Abbildung 41: Piktogramme der Navigation: a) Walk-Modus b) Fly-Modus.

[10] Das Wort Piktogramm setzt sich aus dem lateinischen „pictus" (Bild) und dem griechischen „gramm" (Geschriebenes) zusammen. Wörtlich übersetzt bedeutet es folglich „geschriebenes Bild".

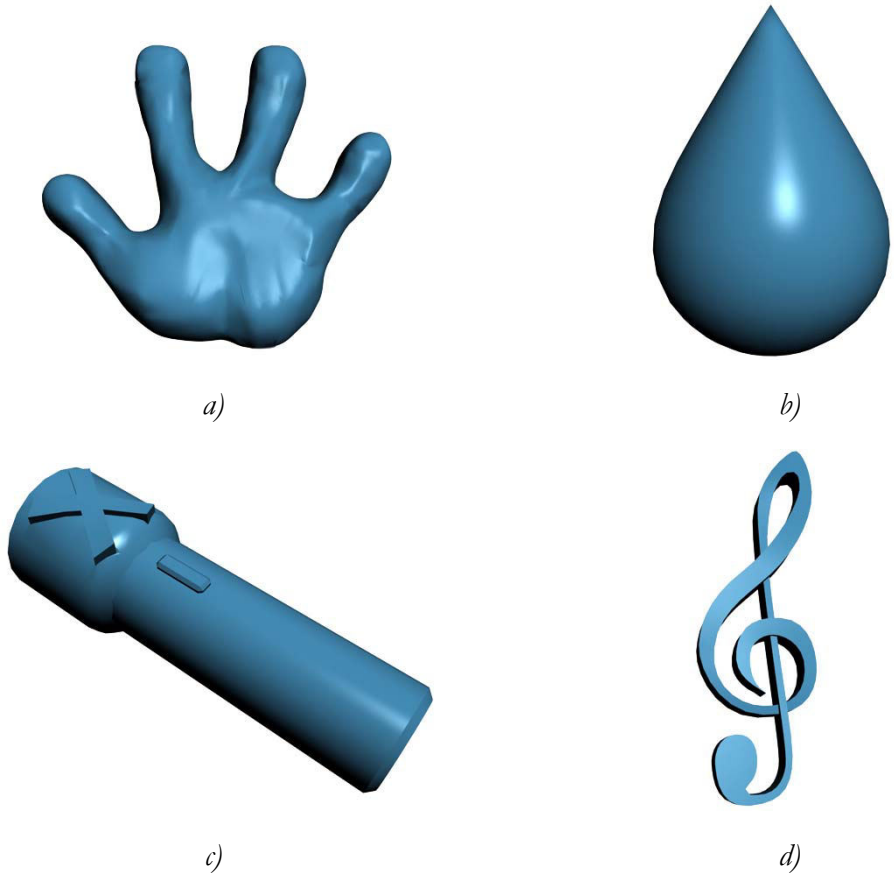


Abbildung 42: Piktogramme der Interaktion: a) Greifen b) Wasseranimation c) X-Ray Taschenlampe d) Orgelmusik.

Zusätzlich sind alle Stellen in der Kirche, an denen der Anwender interagieren oder sich informieren kann, mit einem Piktogramm versehen. Dieser Infopunkt (s. Abbildung 43) hilft dem Anwender dabei, die interessanten Stellen bei einer eigenständigen Erkundung der Kirche zu entdecken und somit nicht ahnungslos an ihnen vorbeizugehen. Die Infopunkte erscheinen nur dann, wenn der Betrachter sich in einem bestimmten Radius um das Objekt befindet und heben sich zur besseren Erkennung von der virtuellen Stiftskirche ab.

Abbildung 43: Piktogramm der informationsbehafteten Punkte.

6.5 Licht in der Stiftskirche

Der realistische Eindruck der Stiftskirche ist abhängig von dem Licht, welches die Szene erleuchtet. Lightning unterstützt die Standard OpenGL-Lichter wie Punktlicht, Spotlicht oder flächiges Licht. Diese lassen sich durch ihre Parameter steuern, um eine angenehme Stimmung in der Stiftskirche zu erreichen. Die definierten Lichter haben ihre Wirkung auf alle Modelle in der Szene. Geht der Anwender nun allerdings in die Gruft, so soll das Licht etwas düsterer in seinem Eindruck werden. Damit dies jedoch nicht zu abrupt geschieht und somit wie ein Darstellungsfehler wirkt, muss das Licht, während sich der Betrachter in die Gruft bewegt, langsam dunkler werden. Geht er aus der Gruft zurück in das Kirchenschiff, so muss das Licht wieder heller werden, ohne dass der Benutzer irritiert wird. Diese Art der Lichtsteuerung lässt sich natürlich auch in allen weiteren Bereichen des VE anwenden (z.B. wenn der Betrachter sich außerhalb der Stiftskirche oder im Turm befindet), um einen charakteristischen Helligkeitseindruck zu erzeugen.

7 Implementierung der einzelnen Objekte

Die beschriebenen Anforderungen und Funktionen sollen nun in einer Applikation zusammengefasst werden. Dazu werden die Funktionen in einzelne [incrTcl]-Objekte (ab hier sind mit Objekten immer die Objekte in der Softwareprogrammierung gemeint) gekapselt und von einem zugehörigen Controller, der wiederum selbst ein Objekt darstellt, kontrolliert und angesteuert. In einem Tcl-Skript, welches die Applikation startet, werden die einzelnen [incrTcl]-Objekte instanziert.

Der Anwender benötigt zur Interaktion und Navigation nur ein Eingabegerät sowie zwei sich daran befindende Buttons. Mit Button2, welchen er mit dem Daumen steuern kann, werden vier verschiedene Modi geschaltet, die nacheinander aktiviert werden. Die ersten zwei Modi, Gehen und Fliegen, stehen immer zu Verfügung, während der Interaktions- und Informationsmodus nur dann als weitere Auswahlmöglichkeiten bereitgestellt werden, wenn sich ein interagierbares oder informatives Modell in Reichweite befindet. Die Aktion kann der Anwender schließlich mit dem Zeigefinger an Button1 direkt ausführen.

Um auf die Abbildung von speziellem Programmcode weitgehend verzichten zu können, wird die Funktionsweise und Struktur der programmierten [incrTcl]-Objekte in Struktogrammen nach der Konvention von Nassi-Schneiderman-Diagrammen erläutert. Alle Klassen der Applikation sind auf der beigefügten CD zu finden.

7.1 Navigationselemente

Die Aktion der Fortbewegung soll ausgeführt werden, wenn sich der Anwender im Modus Walk oder Fly befindet, Button1 drückt und dabei das Eingabegerät in die gewünschte Richtung bewegt. Im Folgenden sind die zur Fortbewegung notwendigen Softwarekomponenten ausführlich in ihrer Funktion beschrieben.

7.1.1 Die Vaterklasse clNavigation

Wie bereits in der Konzeption erläutert, haben die Bewegungsmetaphern Fly und Walk eine große Ähnlichkeit, da sie sich nur durch ihre unterschiedliche Anzahl an Freiheitsgraden unterscheiden. Somit bietet es sich an, eine allgemeine Vaterklasse zu entwickeln und die Objekte für Walk und Fly davon abzuleiten und zu spezialisieren.

Die [incrTCL]-Klasse fungiert dabei als Container für Lightning-Objekte und dient der Beschreibung von Funktionalität (z.B. Berechnungen) zwischen den in der Applikation verwendeten Lightning-Objekten. Damit ist [incrTCL] ein sprachliches Mittel, welches dabei hilft, die Lightning-Objekte in einen bestimmten Kontext zu bringen, ohne dass Lightning die [incrTCL]-Objekte überhaupt kennt.

Die in der Klasse definierten Ein- und Ausgabefelder der Objekte dienen dazu, Routen von anderen Objekten auf diese Felder legen zu können, um den Aufbau eines Datenflusses zu gewährleisten. Die Konvention der Felddefinition lautet:

{ [Feldname] [Datentyp] [Ein/Ausgabe] }.

Das Objekt der Vaterklasse clNavigation besitzt die folgenden Felder:

```
{handPosIn vec3 In}        # Getrackte Position des Eingabegerätes
{handOriIn vec3 In}        # Getrackte Orientierung des Eingabegerätes
{mouseLeftIn int In}       # Eingabegerät Button1 gedrückt
{positionIn vec3 In}       # Aktuelle Position des Viewpoint
{orientationIn vec3 In}    # Aktuelle Orientierung des Viewpoint
{positionOut vec3 Out}     # Neu berechnete Position des Viewpoint
{orientationOut vec3 Out}  # Neu berechnete Orientierung des Viewpoint
```

Die Berechnung der neuen Position erfolgt folgendermaßen: Ist der Modus auf Fly oder Walk gesetzt sowie Button1 gedrückt und gehalten, werden die Daten aus den Feldern handPos und handOri einmalig in *x*-, *y*- und *z*-Variablen gespeichert, welche so lange existieren, bis der Anwender den Button wieder loslässt. Dadurch können die Änderungen der Position, die der Anwender mit dem Eingabegerät ausführt, berechnet werden (Delta).

Die aktuelle Position und Orientierung wird für jede Achse im Raum einzeln in eine Variable gespeichert. Danach wird die Berechnung ausgeführt und anschließend wieder in einem dreidimensionalen Vektor (vec3) für die neue Position und Orientierung zusammengesetzt und in das Ausgabefeld geschrieben. Diese Berechnung beruht auf Kosinus- und Sinusfunktionen der aktuellen Position mit den über das Input Device eingegebenen Positionen und Orientierungen. Zusätzlich gehen Konstanten für die Steuerung der Schnelligkeit (speed) und des Lenkwinkels (angleSpeed) in die Berechnung ein. Das folgende Struktogramm soll den beschriebenen Sachverhalt nochmals verdeutlichen.

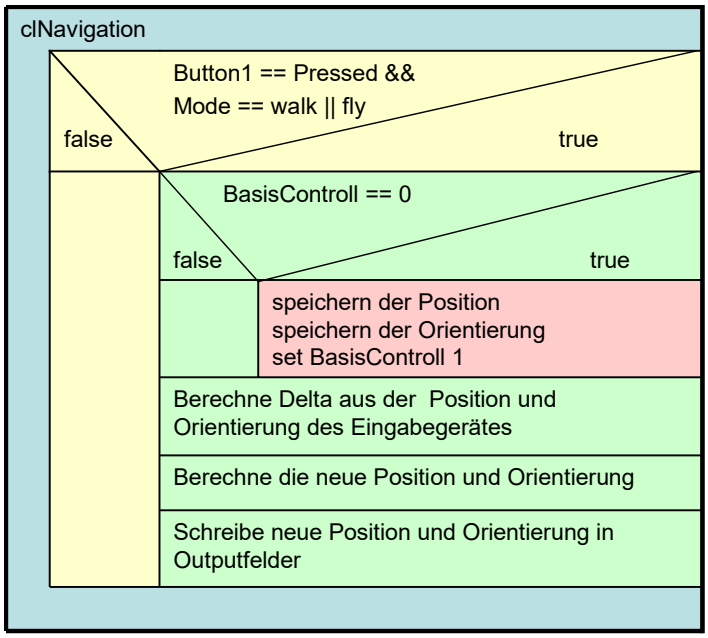


Abbildung 44: Funktionsweise von Navigation.

Im Folgenden soll anhand eines Beispiels der Klasse clNavigation, welche zur Verbesserung der Übersichtlichkeit jedoch stark vereinfacht wurde, gezeigt werden, wie die in [incrTCL] programmierten Objekte aufgebaut sind und die gewünschte Funktionalität implementiert werden kann. Zudem soll dadurch verdeutlicht werden, wie das Objekt strukturiert ist und die einzelnen Lightning-spezifischen Objekte (wie ein ltscript) in das Softwareobjekt integriert werden.

```
01 class Navigation {
02 protected variable iname ""
03    constructor {value} {
04       navigation $value
05    }
06
07 method navigation {value} {
08    set iname [string map {:: _} $this ]
09    set angleSpeed 0.1
10    set speed 0.25
11
12    ltscript navigation$iname {
13       {handPosIn vec3 In}
14       {handOriIn vec3 In}
15       {mouseLeftIn int In}
16       {positionIn vec3 In}
17       {orientationIn vec3 In}
18       {positionOut vec3 Out}
19       {orientationOut vec3 Out}
20    } {
21       proc deg2rad { valueInput } {
22          set output [expr ($valueInput / 360) * 2 * 3.14159 ]
23          return $output
24       }
```

```
25
26        # new position & orientation
27        if {[ltget mouseLeftIn]==1 && ([ltgetA modeSelector -modeOut]
28            == walk || [ltgetA modeSelector -modeOut] == fly)} {
29           scan [ltget orientationIn] "%f %f %f" xori yori zori
30           scan [ltget positionIn] "%f %f %f" xpos ypos zpos
31           scan [ltget handOriIn] "%f %f %f" xhori yhori zhori
32           scan [ltget handPosIn] "%f %f %f" xhpos yhpos zhpos
33
34           # new orientation
35           set xori [expr $xori + ($xhori * $angleSpeed)]
36           set yori [expr $yori + ($yhori * $angleSpeed)]
37           set zori [expr $zori + ($zhori * $angleSpeed)]
38           ltset orientationOut "$xori $yori $zori"
39
40           # new position
41           set ypos [expr $ypos * $speed * cos([deg2rad $xori]) *
42                    cos([deg2rad $yori])]
43           set xpos [expr $xpos * $speed * sin([deg2rad $xori]) *
44                    cos([deg2rad $yori])]
45           set zpos [expr $zpos + $yhposDelta * $speed *
46                    sin([deg2rad $yori])]
47
48           ltset positionOut "$xpos $ypos $zpos"
49        }
50     }
51 }
52
53 method getPositionIn {} {
54     return "navigation$iname positionIn"
55 }
56 method getPositionOut {} {
57     return "navigation$iname positionOut"
58 }
59 method getOrientationIn {} {
60     return "navigation$iname orientationIn"
61 }
62 method getOrientationOut {} {
63     return "navigation$iname orientationOut"
64 }
65 method getHandPosIn {} {
66     return "navigation$iname handPosIn"
67 }
68 method getHandOriIn {} {
69     return "navigation$iname handOriIn"
70 }
71 method getMouseLeftIn {} {
72     return "navigation$iname mouseLeftIn"
73 }
74}
```

Die Zeilen 01 bis 05 zeigen das Erstellen des Klassenrumpfs der Klasse Navigation. Der bei der Instanzierung übergebene Parameter (value) wird im Konstruktor zu dem Objektnamen hinzugefügt.

In Zeile 07 wird die Methode Navigation erzeugt. Innerhalb dieser Methode werden zunächst die verwendeten Variablen deklariert und mit den gewünschten Werten vorinitiali-

siert (Zeile 08-10). Anschließend wird in Zeile 12 ein Lightning Script erzeugt, in welchem die Ein- und Ausgabefelder mit ihren Datentypen definiert werden (Zeile 13-19). Die Logik des Navigations-Objekts strukturiert sich auf die folgende Art und Weise. Zuerst ist eine Prozedur (deg2rad in Zeile 21-24) definiert, welche Werte aus dem Gradmaß in das Radialmaß umrechnet. Anschließend kommt die Bedingung (if-Abfrage), bei deren Erfüllung die folgende Funktionalität ausgeführt wird.

Die einzelnen Werte der Vektoren von Position und Orientierung des Eingabegerätes sowie des aktuellen Standortes werden in einzelne *x*-, *y*- und *z*-Variablen geschrieben, wie in Zeile 29-32 zu sehen ist. Aus diesen einzelnen Werten der Orientierungen kann mit den in Zeile 35-37 dargestellten Gleichungen die neue Orientierung berechnet werden. Mit der Prozedur deg2rad werden die Werte von Grad- nach Radialmaß umgerechnet. Durch den in der Variable anglespeed angegebenen Wert ist die Geschwindigkeit der Orientierungsänderung steuerbar. In Zeile 38 werden die hierbei berechneten Werte zurück in einen Vektor gespeichert und in das Ausgabefeld orientationOut geschrieben.

In Zeile 41-46 findet ein ähnlicher Vorgang statt, durch welchen aus der Position des aktuellen Standortes sowie aus der Position und Orientierung des Eingabegerätes eine neue Position für die jeweiligen *x*-, *y*- und *z*-Werte errechnet wird. Diese werden dann (Zeile 48) in einen Vektor geschrieben und an das Feld positionOut übergeben.

In den Zeilen 53-73 sind die einzelnen Methoden des Objekts aufgeführt. Diese dienen hier der Verbindung der im ltscipt navigation angegebenen Felder, mit den Feldern anderer Objekte mittels Routen. Wird zum Beispiel die Methode getPositionOut des Objekts aufgerufen, so wird der Wert aus dem Feld positionOut ausgelesen und an den Aufrufer übergeben.

Die Erstellung einer Route von einem Objekt (hier beispielsweise das Objekt Pathfinder) zum Navigations-Objekt sieht demnach folgendermaßen aus:

```
ltroute "[pathfinder getPositionIn]" "[navigation getPositionOut]"
```

Auch Routen von einem [incrTCL]-Objekt zu einem Lightning-Objekt sowie Routen zwischen Lightning-Objekten sind möglich, wie der folgende Ausdruck veranschaulicht.

```
ltroute "[navigation getPositionOut]" "camera positionIn"
```

Der vollständige Code der Klasse Navigation sowie aller zur Applikation gehörenden Klassen ist, wie bereits erwähnt, auf der im Anhang dieser Arbeit beigefügten CD zu Verfügung gestellt.

7.1.2 Die Kindklasse clFly

Die Klasse clFly bekommt jegliche Funktionalität von der Vaterklasse clNavigation vererbt, spezialisiert sich jedoch, indem sie die *z*-Orientierung immer auf den Wert 0 zwingt. Damit ist der Freiheitsgrad der Orientierung eingeschränkt. Zusätzlich wird die Geometrie des Piktogrammfliegers (Flieger.pfb), welche mit einer öffentlichen Methode (setVisibleFly) sichtbar oder unsichtbar gemacht werden kann, an dieser Stelle geladen.

7.1.3 Die Kindklasse clWalk

Auch an die Klasse clWalk wird alle Funktionalität der Vaterklasse vererbt. Die Spezialisierung ist hier jedoch um einiges umfangreicher. Zunächst werden die DOF der Orientierung beschränkt, sodass nur die *x*-Orientierung (also das Drehen des Kopfes) zum Tragen kommt. Des Weiteren wird für den „terrain follow"-Algorithmus ein zusätzliches Objekt, ein ltSelectRay, welches das Framework Lightning bereithält, eingefügt. Der Selectray erhält dieselben Positionen wie der Viewpoint und ist parallel zur *z*-Achse nach unten gerichtet. Er bewegt sich also mit dem Betrachter über den Boden und tastet diesen dabei ab. Der Selectray gibt immer den Punkt an, an dem er eine Geometrie durchdringt. Von diesem Punkt wird nur den *z*-Wert benötigt, zu welchem dann 185 Zentimeter hinzuaddiert werden. Dieser Wert wird als neue Position im *z*-Parameter in das Ausgabefeld, welches mit dem aktuellen Viewpoint verroutet ist, geschrieben. Dadurch entsteht bei dem Benutzer der Eindruck, sich gehend zu bewegen, da er in einem definierten Abstand über den Boden gleitet. Die Logik des Terrain following funktioniert auch bei Treppenstufen oder schiefen Ebenen, solange die Geometrien, welche der Selectray abtastet, keine offenen Stellen oder Risse beinhalten. Zusätzlich wird die Geometrie der Piktogrammschuhe geladen, welche mit dem Aufruf einer öffentlichen Methode (setVisibleWalk) sichtbar oder unsichtbar gemacht werden kann.

Auf Diagramme zu den Klassen Walk und Fly wird an dieser Stelle verzichtet, da diese bis auf wenige Stellen äquivalent zu der Vaterklasse sind.

7.1.4 Die Klasse clMotionController

Instanzierte Objekte der Navigation (Walk und Fly) werden durch den MotionController kontrolliert und angesteuert. Da zu einem Zeitpunkt jeweils nur eine Route von Fly oder Walk in Richtung des Kamera-Objekts führen darf, steuert der MotionController die Routen in Abhängigkeit des ausgewählten Modus zu den einzelnen Objekten und der Kamera.

Des Weiteren steuert er die Sichtbarkeit der Piktogramme von Walk und Fly, damit der Anwender immer das richtige Feedback zum aktiven Status erhält.

Das Objekt benötigt nur die folgenden zwei Eingabefelder:

```
{mouseLeftIn int In}    # Eingabegerät Button1 gedrückt
{modeIn int In}         # Aktuell ausgewählter Modus
```

Alle weiteren Funktionen werden anhand von Methodenaufrufe gesteuert.

Wählt der Anwender den Modus zum Fliegen aus, so werden Routen (für Position und Orientierung) vom Fly-Objekt in Richtung Kamera aufgebaut. Sind schon Routen vom Walk-Objekt vorhanden, werden diese zuerst gekappt. Nun ist gewährleistet, dass alle Eingaben des Anwenders zur Änderung des Viewpoint über das Fly-Objekt berechnet und an die Kamera weitergegeben werden. Zusätzlich wird das Piktogramm des Fliegers angezeigt. Geht der Anwender aber einen Modus weiter und aktiviert den Walk-Modus, werden wiederum alle Routen zum Kamera-Objekt gekappt und neue Routen für die Position und Orientierung vom Walk-Objekt in Richtung des Kamera-Objekts aufgebaut. Ist jedoch der Interaktions- oder Informationsmodus aktiviert, haben die Navigationsobjekte keinen Einfluss mehr. Somit wird auch keine weitere Änderung an der aktuellen Position und Orientierung ausgeführt. Der Programmablauf ist im folgenden Diagramm veranschaulicht.

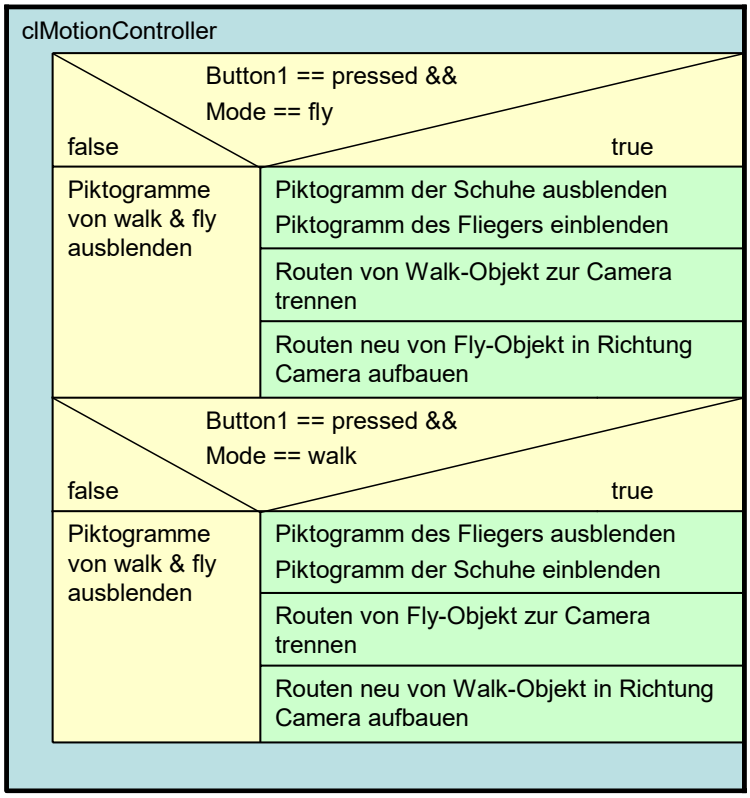


Abbildung 45: Funktionsweise des MotionController.

7.2 Interaktion mit den Modellen

Bei der Implementierung des Interaktionsmodells sind die einzelnen Interaktionsmöglichkeiten in Objekte gekapselt und werden von einem InteraktionsController gesteuert und verwaltet. Im Folgenden werden die einzelnen Objekte in ihrer Implementation und Funktion beschrieben.

7.2.1 Die Klasse clGrabber (die Greiflogik)

Das Objekt, welches das Greifen des auf dem Altar liegenden Buches ermöglicht, verwendet mehrere von Lightning bereitgehaltene Zusatzelemente. Zuerst wird die Geometrie der Piktogrammhand geladen. Zusätzlich erzeugt man einen ltSelectRay und ein Gruppenelement (ltGroup[11]). Auf jedes dieser Elemente werden Position und Orientierung des Eingabegerätes geroutet. Der Selectray erhält eine bestimmte Ausrichtung und Größe, um ihn in der Piktogrammhand verschwinden zu lassen. Des Weiteren sensibilisiert man ihn, damit er nur auf die Geometrie des Buches reagiert. Wenn der Anwender nun vor dem Altar steht, den Interaktionsmodus auswählt und mit der Piktogrammhand in die Geometrie des Buches eindringt (der Selectray muss dabei die Geometrie des Buches berühren), ist es durch die Betätigung des Button1 möglich, das grafische Element des Buches in die Gruppe einzufügen. Der Anwender kann das Buch beliebig bewegen und drehen, da die Parameter der Position und Orientierung des Eingabegerätes über die Gruppe an das Buch weitergegeben werden. Somit hat der Anwender den Eindruck, das Buch in der Hand zu halten. In dem Augenblick, in dem er das Buch greift, wird die Piktogrammhand annähernd unsichtbar, wodurch ein ungehinderter Blick auf das Buch entsteht. Sobald er den Button1 loslässt, wird die Geometrie wieder aus der Gruppe genommen und an den Wurzelknoten der Szene gehängt. Zusätzliche Elemente wie x-Form Transformationen zur Umrechnung der Positionen und Orientierungen sind fest miteinander verroutet. Daher benötigt dieses Objekt nur die folgenden Eingabefelder zum Aufbau des Datenflusses.

```
{modeIn int In}         # Aktuell ausgewählter Modus
{mouseLeftIn int In}    # Eingabegerät Button1 gedrückt
{hitCountIn int In}     # Anzahl von Selectray getroffener Objekte
{nodeHitIn string In}   # Name des vom Selectray getroffenen Objekts
```

Die folgende Abbildung zeigt die erklärten Funktionen in einem Struktogramm.

---

[11] Das Element der ltGroup ist ein Container für grafische Elemente. Objekte können nach Belieben hinzugefügt oder herausgenommen werden. Die Position und Orientierung der Gruppe wird relativ auf die beinhalteten Objekte übertragen.

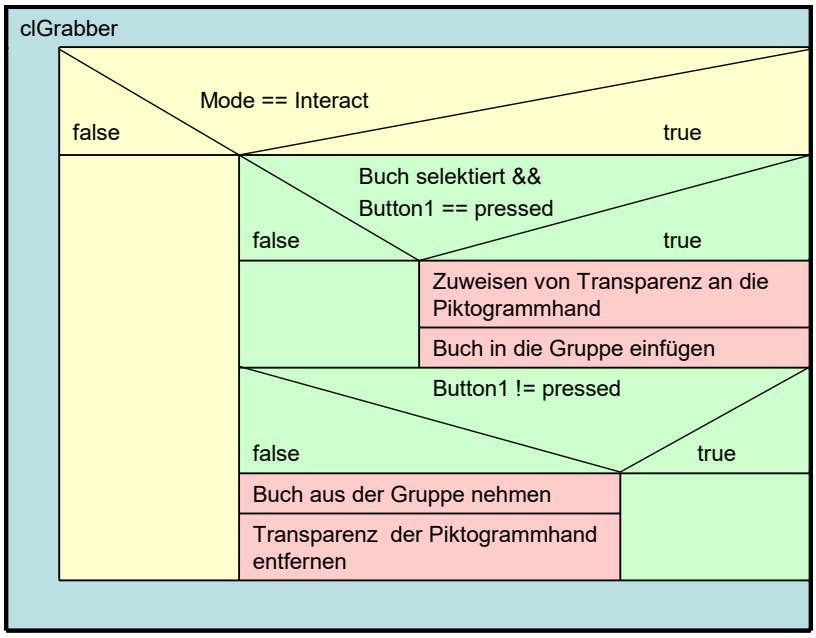


Abbildung 46: Funktionsweise des Grabber.

7.2.2 Die Klasse clWaterReactor

Das Objekt des WaterReactors ermöglicht es, mit dem Wasser im Taufbecken zu interagieren und auf der Wasseroberfläche eine Welle zu erzeugen. Auch hier werden mehrere von Lightning bereitgehaltene Zusatzelemente verwendet. Zuerst wird (wie bei jedem Interaktionsobjekt) der zugehörige graphische Repräsentant, hier der Wassertropfen, geladen. Anschließend wird wieder einen ltSelectRay erzeugt, welcher eine solche Ausrichtung und Größe erhält, dass er in dem Piktogrammwassertropfen verschwindet. Schließlich sensibilisiert man ihn mit der Wasseroberfläche. Wassertropfen und Selectray bekommen Position und Orientierung ebenfalls vom Eingabegerät durch Routen übermittelt. Ist der Interaktionsmodus ausgewählt und die Position des Anwenders am Taufbecken, kann der Anwender eine Welle erzeugen, indem er die Geometrie der Wasserfläche mit dem Piktogrammwassertropfen berührt.

Die Animation des Wassers funktioniert durch die Verwendung eines dynamischen Geometrieobjekts (ein in C++ geschriebenes Lightning-Objekt) als Wasserfläche. Um ein Ausbreiten der Wellen zu simulieren, wird die Funktion einer gedämpften Schwingung verwendet, die zu jedem Punkt eines Polygons (Dreieck) einen entsprechenden *z*-Wert und die zugehörige Normale liefert. Das bedeutet, die Animation iteriert über jedes Polygon und berechnet den entsprechenden *z*-Wert anhand der Schwingungsfunktion und des Abstandes des Punktes zum Berührungspunkt. Die verwendete Funktion der gedämpften Schwingung generiert ein Wellenmodell, das am Berührungspunkt mit hoher Amplitude

startet und mit zunehmendem Abstand zum Ursprung die Amplitude der Welle verringert (siehe dazu auch [del00] S.187-194).

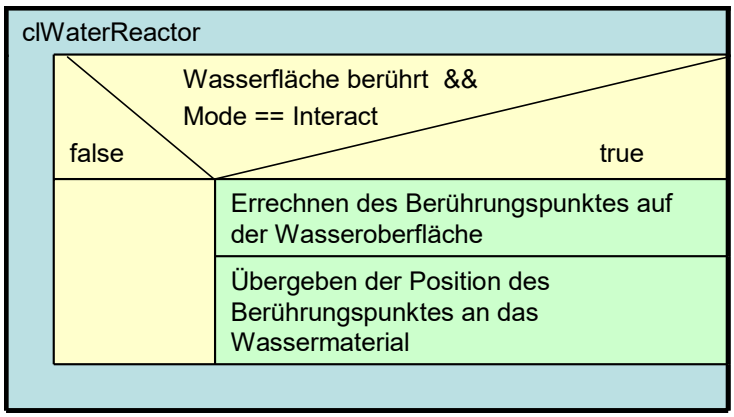


Abbildung 47: Funktionsweise des WaterReactor.

7.2.3 Die Klasse clSargReactor (X-Ray Taschenlampe)

Das Objekt verhält sich ähnlich wie die bisher erklärten Objekte. Wenn der Anwender sich am Sarg befindet, kann er mit dem Eingabegerät eine Aktion auslösen. An dieser Stelle wird die bereits erwähnte X-Ray Taschenlampe eingesetzt und das entsprechende Piktogramm visualisiert. Erweitert wird die Interaktionsmöglichkeit durch den Einsatz eines in Lightning verfügbaren Schnittebenen-Objekts (ltCutplane). Beide Elemente werden wieder über Routen mit dem Eingabegerät verbunden, wobei die Schnittebene selbst in einem definierten Abstand vor dem Piktogramm positioniert ist. Wenn der Anwender nun Button1 drückt, so wird die Schnittebene aktiviert. Der Anwender kann jetzt die Ebene in allen 6 Freiheitsgraden verändern und erhält dabei den Eindruck, eine Taschenlampe in der Hand zu halten, mit welcher er die Wand des Sarges durchleuchten kann.

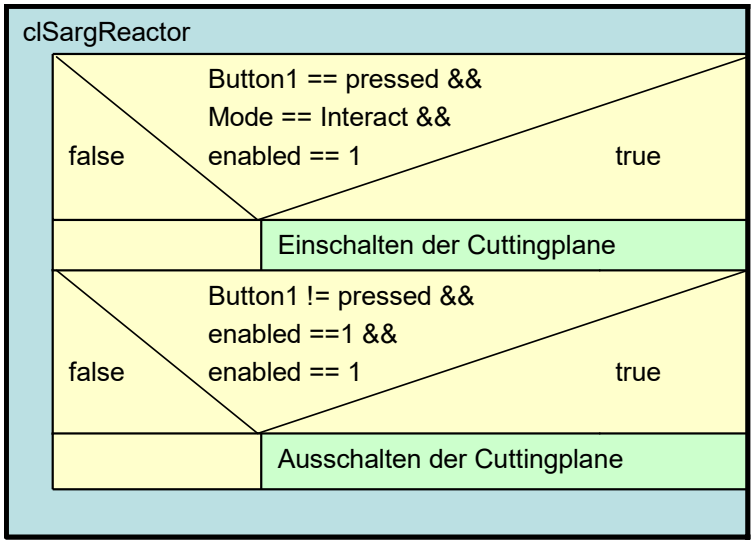


Abbildung 48: Funktionsweise des SargReactor.

Die Durchleuchtungsfunktion der ltCutplane wird durch den Einsatz einer OpenGL Schnittebene (Clipping plane) und eines Stencil-Buffers [yio01] realisiert. Das Wörtchen 'Buffer' verrät bereits, dass der Stencil-Buffer im Grunde genommen ein weiterer, für den Grafikchip nutzbarer Speicherbereich ist, wie der *z*-Buffer für Tiefenwerte und der Framebuffer für Farbwerte. In der Tat werden die Werte des Stencil-Buffers zusammen mit den Tiefenwerten gespeichert, wobei die Tiefenwerte die ersten 24 bits eines Wortes (32bit Packet) und die Werte des Stencil-Buffers die übrigen 8 bits belegen.

'Stencil' bedeutet übersetzt Schablone oder Maskierung, was darauf schließen lässt, dass bestimmten Bereichen des Bildes, sogar einzelnen Pixel, bestimmte Werte zugewiesen werden kann. Bildlich gesprochen hängt an jedem Pixel ein kleiner Notizzettel, auf dem sich der 3D-Chip bestimmte Dinge "notieren" kann. Der 1-bit Stencil-Buffer ermöglicht dabei zwei Zustände, 1 oder 0, was in unserem Fall sichtbar oder unsichtbar bedeutet [mau99].

Übertragen auf die X-Ray Taschenlampe bedeutet dies, dass eine Schnittebene durch das Modell des Sarges gelegt wird. Die Polygone der Geometrie werden gegen die Schnittebene getestet und an den Schnittstellen gekappt. Dies ist eine OpenGL-Operation und wird direkt in der Hardware der Grafikkarte ausgeführt. Beim Rendervorgang werden die Werte, welche in den Stencil-Buffer geschrieben werden, an jeder Stelle, an der die Schnittebene ein Polygon berührt, invertiert. Damit wird im Stencil-Buffer eine Maske aufgebaut, deren mit 1 markierte Stellen das Innere und mit 0 markierte Stellen das Äußere des Objekts repräsentieren. Die durch den Schnittvorgang offenen Stellen in der Geometrie sind im Stencil-Buffer mit 1 gekennzeichnet und werden zum Schluss durch ein Polygon (Cap) über die ganze Fläche geschlossen. Anschließend wird die gewünschte Textur des Holzsarges auf diese Flächen gezeichnet. Unerlässlich ist hierbei, dass die Geometrie des Objekts geschlossen ist und kein Backfaceculling verwendet wird, denn sonst ist die korrekte Funktion des Stencil-Algorithmus nicht gewährleistet und die Schnittfläche der Geometrie kann nicht korrekt mit dem „Cap“ geschlossen werden. Die folgende Abbildung soll den beschriebenen Vorgang anhand einer einfachen Beispielgeometrie eines ausgehöhlten Würfels verdeutlichen.

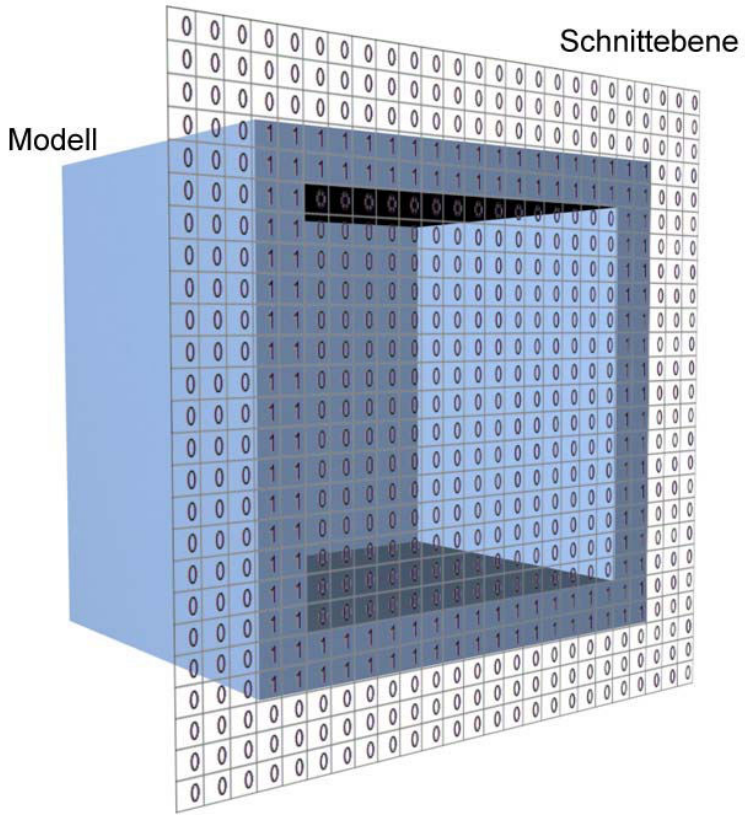


Abbildung 49: Funktionsweise der X-Ray Taschenlampe.

7.2.4 Die Klasse clOrgelReactor (Aktivieren des Sounds)

Dieses Objekt ist ähnlich strukturiert wie das Objekt des Taufbeckens. Als Piktogramm kommt hier der Notenschlüssel zum Einsatz, an welchem ein SelectRay befestigt ist, auch diese Elemente erhalten wieder die Position und Orientierung des Eingabegerätes. Zusätzlich werden zwei Musikstücke geladen. Zum einen ein ambientes Musikstück als Hintergrundmusik, welches in einer Endlosschleife abgespielt wird, und zum anderen ein Orgelstück von Bach, welches in dem Augenblick abgespielt wird, in dem der Anwender eine der Orgelpfeifen oder den Kasten der Orgel berührt und den Vorgang mit Button1 aktiviert. Um dem Anwender die Aktivierung des Musikstücks zu visualisieren, wird das Piktogramm des Notenschlüssels transparent.

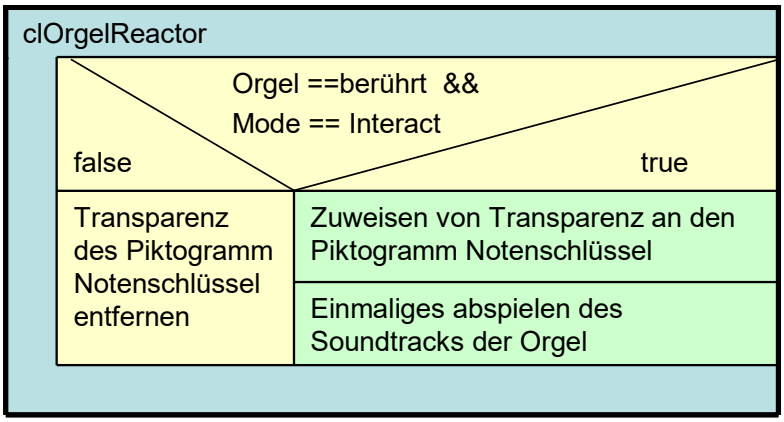


Abbildung 50: Funktionsweise des OrgelReactor.

7.2.5 Die Klasse InteractionController

An dieser Stelle benötigt man den InteractionController, ein Objekt, welches die einzelnen Interaktionsobjekte steuert und kontrolliert. Dieser berechnet, solange der Anwender sich in der virtuellen Umgebung befindet, kontinuierlich die Abstände des Anwenders zu den einzelnen interagierbaren Modellen in der Szene. Für jedes Modell wurde ein bestimmter Aktionsradius definiert. Dieser beträgt am Altar zum Beispiel 1,5 Meter. Bewegt der Anwender sich innerhalb des Radius, kann er mit dem Modusumschalter Button2 in den Interaktionsmodus wechseln. Das System stellt ihm nun automatisch die zugehörige Aktionsmöglichkeit (z.B. Greifen) bereit und er kann direkt mit dem Modell interagieren. Damit der Anwender erkennt, welche Art der Interaktion an den Modellen jeweils möglich ist, wird das entsprechende Piktogramm angezeigt. Befindet der Anwender sich jedoch nicht innerhalb eines Aktionsradius, kann er auch keine Interaktionen ausführen und wird daher automatisch einen Modus weiter zum Informationsmodus befördert.

Um die notwendigen Berechnungen der Funktionen auszuführen, benötigt das Objekt bestimmte Felder:

```
{positionIn vec3 In}    # Aktuelle Position des Viewpoint
{orientationIn vec3 In} # Aktuelle Orientierung des Viewpoint
{mouseLeftIn int In}    # Eingabegerät Button1 gedrückt
{modeIn int In}         # Aktuell ausgewählter Modus
```

Folgende Abbildung zeigt das Objekt des InteractionController als Struktogramm.

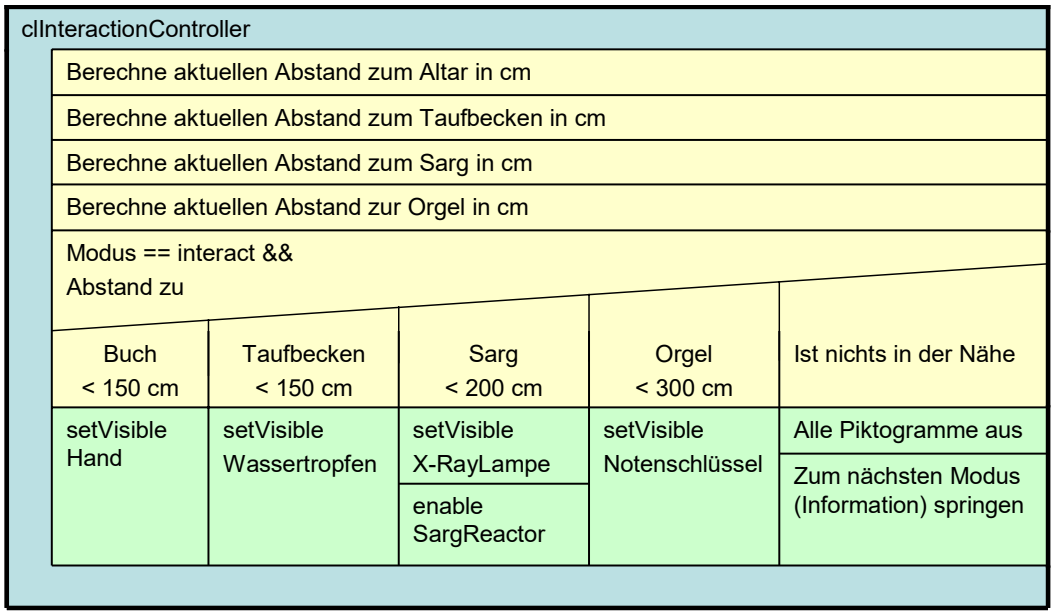


Abbildung 51: Funktionsweise des InteractionController.

7.3 Informations-Visualisierung

7.3.1 SQLite Datenbank

Die Ablage der Informationen zu den in Kapitel 5.1 vorgestellten informativen Modellen findet in der Datenbank SQLite statt. Hierfür ist nur eine Tabelle (tblAnnotatio) notwendig, deren Aufbau in Abbildung 52 dargestellt wird. Die Datenbank besteht aus einer einzelnen Datei, wobei unbedingt zu beachten ist, dass diese sich nicht auf einem Netzlaufwerk, sondern auf einem lokalen Verzeichnis befindet. In dieser Arbeit hat sie ihren Platz auf allen Knoten im Clustersystem unter /tmp.

Eingebunden wird die Datei tclsqlite.so mit dem folgenden Load-Befehl, wodurch die Tcl Syntax um einen weiteren Befehl (sqlite) erweitert wird.

```
load /*/sqlite/tclsqlite-2.8.15.so sqlite
```

Des Weiteren wird mit der folgenden Zeile eine Verbindung zu der sich im Verzeichnis /tmp befindenden Datenbank StiftskircheInfoDB hergestellt.

```
sqlite infoDB /tmp/StiftskircheInfoDB
```

	Identifier	Layer	Previous	Next	Content	Date
1	Brenz_Epitaph_...	2	Brenz_Epitaph_...	Brenz_Epitaph_...	Unterhalb steh...	
2	Brenz_Epitaph_...	3	Brenz_Epitaph_...		beigestanden d...	
3	Grabplatte_1	1		Grabplatte_2	Ludwig Naucler...	
4	Grabplatte_2	2	Grabplatte_1	Grabplatte_3	die Magisterwu...	
5	Orgel_1	1		Orgel_2	Schon im Jahre...	
6	Orgel_2	2	Orgel_1	Orgel_3	diese legendae...	
7	Standbild_1	1		Standbild_2	Grafenstandbil...	
8	Standbild_2	2	Standbild_1		Wappenfeldern ...	
9	Gruft_1	1		Gruft_2	Grablege des H...	
10	Gruft_2	2	Gruft_1		eine Verbindun...	
11	Eingang_1	1		Eingang_2	Die Stiftskirc...	

Abbildung 52: Aufbau der Tabelle tblAnnotation.

7.3.2 Die Klasse clInfoMenue

Die Schnittstelle zwischen der Information und dem Anwender liefert hier eine Klasse namens clInfoMenue. Diese ist zuständig für die Darstellung der Inhalte und die Bereitstellung zusätzlicher Eingabemöglichkeiten für den Benutzer. Das daraus instanzierte Objekt ist also ein Userinterface mit vorgegebenen, vom Inhalt unabhängigen Ein- und Ausgabemöglichkeiten. Zum einen ist eine Textplane für die Anzeige der Information vorhanden und es wurden Steuerelemente für das Blättern innerhalb einer Information (eine Seite weiter; eine Seite zurück) angebracht. Zum anderen befindet sich hier, in Form einer Schaltfläche, die Aktivierung der geführten Tour durch die Stiftskirche. Die mit 3D Studio Max modellierte Geometrie des Informationsmenüs beinhaltet eine Ebene zum Aufbringen des Textes sowie die drei Schaltflächen, welche eindeutig benannt wurden. Für die Anzeige des Textes hält Lightning ein eigenes Objekt namens ltText bereit, welchem ein String als Ausgabetext sowie eine bestimmte Schriftart (im Type1 Format) zugewiesen werden kann.

Da sich der Anwender frei durch die Kirche hindurchbewegen kann, ist das InfoMenue mit der Position des Anwenders verbunden. Es wird jedoch nur sichtbar, wenn dieser sich an einem informativen Modell befindet und den Informationsmodus aktiviert. Damit das Interface dem Anwender die Sicht auf das interessante Modell nicht versperrt, kann er es beliebig in dessen relativer Position um sich herum bewegen. Diese Funktionalität wird mit drei ltGroup Objekten realisiert. Eine Gruppe wird mit der Position und Orientierung des Anwenders verbunden und dient somit als Basisgruppe (baseGroup). In diese wird eine weitere Gruppe (middleGroup) eingefügt, welche wiederum eine Gruppe als Container für die zum Menü gehörenden grafischen Objekte und die Textplane (infoBoxGroup) beinhaltet. Bewegt sich der Betrachter nun, so folgen ihm alle Gruppen und somit auch die einzelnen Modelle des InfoMenue. Die Verschachtelung der Gruppen ineinander ist notwendig, um dem Betrachter eine relative Änderung der Menüposition zu ermöglichen. Dazu wird ein weiteres Werkzeug für den Anwender benötigt, welches das Verschieben des Menüs

ermöglicht und die verschiedenen Funktionalitäten zum Blättern oder Aktivieren der geführten Tour gestattet. Dieses Werkzeug besteht wiederum aus einem ltSelectRay, der auf die Geometrie des InfoMenue sensibilisiert ist. Dieser ist mit der Position und Orientierung des Eingabegerätes mittels Routen verbunden und erhält eine sichtbare Linie als Repräsentant in der virtuellen Umgebung. An diesem Zeigestab befindet sich ebenfalls eine Gruppe, die infoRayGroup. Durchdringt der Anwender die Geometrie der Textplane und drückt Button1, wird die infoBoxGroup von der middleGroup in die infoRayGroup übergeben. Dadurch kann sie beliebig positioniert werden. Wenn der Anwender den Button loslässt, wird die infoBoxGroup zurück in die middleGroup gereicht. Bewegt sich der Benutzer, so folgt ihm das Menü in dieser relativen Position. Berührt er mit dem Auswahlwerkzeug aber die Schaltfläche zum Vorwärts- oder Zurückblättern durch die Information und aktiviert dabei Button1, werden die Felder previousInfoOut oder nextInfoOut des Objekts je nach Auswahl auf 1 oder 0 gesetzt. Dasselbe geschieht, wenn der Anwender die Schaltfläche der geführten Tour aktiviert. Dabei wird jedoch das Feld nextPOIOut (next Point of Interest) auf 1 gesetzt.

Die genannten Felder werden schließlich mit weiteren Feldern von anderen Objekten wie dem InformationController oder dem Objekt der geführten Tour (Pathfinder) durch Routen verbunden. Für die korrekte Arbeitsweise benötigt das Objekt die folgenden Felder:

```
{modeIn int In}            # Aktuell ausgewählter Modus
{mouseLeftIn int In}       # Eingabegerät Button1 gedrückt
{nodeHitIn string In}      # Name des vom Selectray getroffenen Objekts
{hitCountIn int In}        # Anzahl von Selectray getroffener Objekte
{previousInfoOut int Out}  # Vorherige Seite der Information anzeigen
{nextInfoOut int Out}      # Nächste Seite der Information anzeigen
{nextPOIOut int Out}       # geführte Tour aktiviert
{enabled int In}           # InfoMenue aktiviert
```

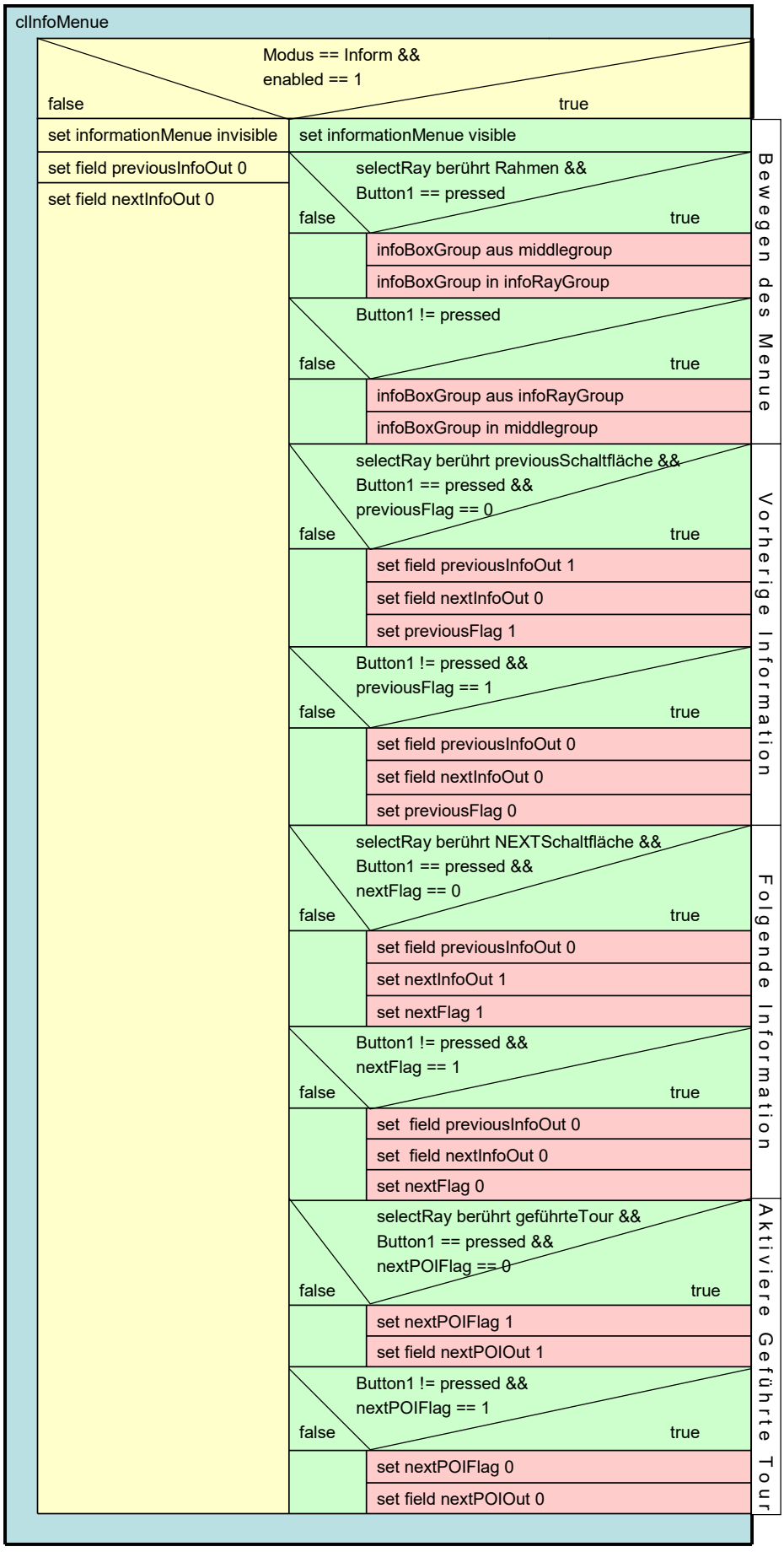


Abbildung 53: Funktionsweise des InformationMenue.

7.3.3 Die Klasse clInformationController

Das im vorherigen Kaptitel dargelegte Userinterface ist mit dem Objekt Information-Controller mittels Routen verbunden. Auf diese Weise erhält der InformationController die Aktionen, welche der Anwender an den Schaltflächen 'previous' oder 'next' des Menüs vornimmt und kann entsprechend darauf reagieren. Des Weiteren ist er für die Anzeige des Infopunktes zuständig, wenn der Betrachter sich in der Nähe eines interessanten Modells bewegt.

Während der Simulation wird der Abstand des Anwenders zu den einzelnen informativen Modellen kontinuierlich errechnet und jeweils in eine Variable gespeichert. Ist die Position zu einem der Modelle kleiner als eine bestimmte Vorgabe, so wird der Infopunkt an diesem Modell eingeblendet. Der Anwender bekommt dadurch signalisiert, dass er sich in der Nähe eines informativen Modells befindet, hat jedoch, solange er nicht nahe genug am Objekt ist, noch nicht die Möglichkeit die Informationen abzurufen. In eine Variable namens actualAnnotation wird der Name des interessanten Punktes, in dessen Nähe er sich gerade befindet, geschrieben (z.B. Grabplatte). Verringert der Anwender schließlich den Abstand zum Modell bis er innerhalb eines definierten Abstandes davor steht, erhält er die Möglichkeit den Informationsmodus zu wählen, wodurch das InfoMenue eingeschaltet und angezeigt wird. In diesem Augenblick wird die Methode dbSelect aufgerufen und der in Variable actualAnnotation stehende String (hier Grabplatte) als Identifier mitgegeben. Die Datenbank gibt nun einen String mit dem zu diesem Identifier gehörenden Inhalt (content) zurück, welcher zur Anzeige an das ltText-Objekt übergeben wird.

Wenn der Anwender die Schaltfläche 'next' mit dem SelectRay des Informations-Modus aktiviert, ändert sich das Feld nextInfoIn des Controllers. Damit wird eine weitere Methode des InfomationControllers aufgerufen (die Methode dbNext), welche eine SQL-Abfrage an die Datenbank übergibt. Diese Abfrage holt den Eintrag zu dieser Information mit dem in actualAnnotation gespeicherten Identifier (Grabplatte) aus der Spalte next (Grabplatte2) und setzt ihn als neuen Identifier in actualAnnotation. Anschließend wird die Methode dbSelect erneut ausgeführt (nun jedoch mit dem neuen Identifier Grabplatte2). Der Betrachter erhält damit den zweiten Eintrag zu der Information an der Grabplatte.

Aktiviert der Anwender die Schaltfläche 'previous', so ist die Funktion analog wie bei 'next'. Allerdings wird die Methode getPrevious aufgerufen, der neue Identifier aus der Spalte previous geholt und in actualAnnotation gespeichert. Mit diesem neuen Identifier wird die Methode dbSelect aufgerufen und der Betrachter erhält die vorherige Information zum Modell.

Ist jedoch keine weitere oder vorherige Information zu dem Objekt vorhanden, so wird NULL als Identifier durch die Abfrage von getNext oder getPrevious zurückgegeben. In diesem Fall wird der alte Wert von actualAnnotation nicht überschrieben und die zuletzt angezeigte Information bleibt bestehen. Der Anwender kann somit nur noch in die jeweils andere Richtung blättern.

Verlässt der Anwender den Radius eines informationsbehafteten Modells, so wird der in ltText angezeigte Inhalt entfernt und das InfoMenue abgeschaltet (disabled) sowie unsichtbar gemacht.

Durch die vorangegangenen Erklärungen sollte deutlich geworden sein, dass der Anwender nur dann Zugriff auf das InfoMenue hat, wenn er sich innerhalb des definierten Radius um ein informatives Modell befindet.

Für die beschriebene Funktionalität benötigt das Objekt folgende Felder:

```
{positionIn vec3 In}       # Aktuelle Position des Viewpoint
{modeIn int In}            # Aktuell ausgewählter Modus
{previousInfoIn int In}    # Vorherige Seite der Information anzeigen
{nextInfoIn int In}        # Nächste Seite der Information anzeigen
{actualAnnotation string Out} # Name des nächstliegenden Modells
```

Die anschließend aufgelisteten Methoden zeigen die in Tcl eingebundene SQL-Syntax für die Abfrage der Informationen aus der Datenbank. Die weiteren Methoden für die Abfrage des nächsten oder vorherigen Identifier funktionieren analog zu dieser.

```
method dbSelect {identifier} {
  set content [infoDB eval "SELECT Content FROM tblAnnotation WHERE
                            Identifier = '$identifier';"]
return $content
}
```

Abbildung 54 zeigt das Funktionsdiagramm des InformationController. Außer der Zeile set actualAnnotation, in welche der jeweilige Name des informativen Modells geschrieben wird, erhalten die mit „…“ gekennzeichneten Felder durchgehend denselben Eintrag.

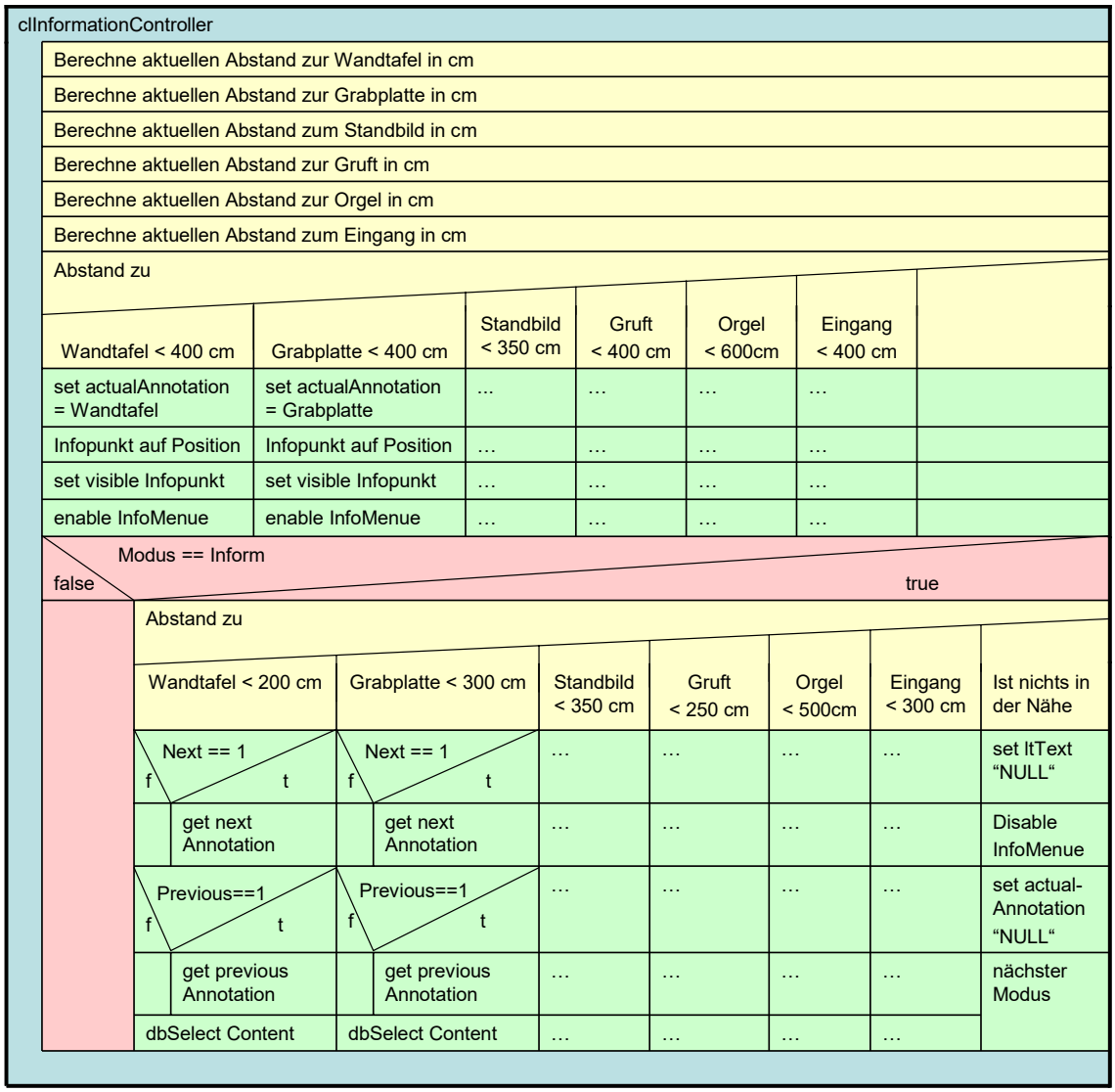


Abbildung 54: Funktionsweise des InformationController.

7.4 Die geführte Tour

Wie bereits erwähnt soll die geführte Tour dem Anwender ein einfaches Besichtigen der Kirche ermöglichen und gewährleisten, dass dieser, auch ohne dass er sich in der Stiftskirche auskennt, alle informativen Modelle findet. Zudem soll er auf Wunsch automatisch und in einer sanften Bewegungsbahn zum nächsten Objekt transportiert werden können. Dafür wurde ein Objekt entwickelt, der Pathfinder, welcher diese Funktionalität bereithält.

7.4.1 Die Klasse clPathfinder

Für die Umsetzung der geführten Tour gibt es mehrere Möglichkeiten. In vielen Computerspielen kommen Wegberechnungsalgorithmen zum Einsatz, die zur Laufzeit einen geeigneten Pfad durch die Geometrie berechnen. Diese Möglichkeit erfordert in verschach-

telten Räumen beispielsweise einen sehr ausgefeilten Algorithmus zur Pfadberechnung und daher, in Abhängigkeit von der Szene, sehr viel Rechenkapazität. Da diese aber hauptsächlich für den Rendervorgang benötigt wird, ist es besser, die Pfade im Vorfeld zu berechnen. In der Stiftskirche besitzen alle informationsbehafteten Modelle eine feste Position, welche sich zur Laufzeit nicht verändert, weshalb die Berechnung eines Pfades nicht notwendig ist. Stattdessen ist es möglich, einen festen Pfad zwischen den Modellen zu definieren, auf dem sich das Kamera-Objekt bewegt. Um zu gewährleisten, dass der Blickwinkel des Anwenders sich in der Bewegungsrichtung orientiert und sein Blick sich auf das interessante Modell richtet, wird dessen Orientierung ebenfalls über einen Pfad beeinflusst. Die genannten Pfade setzen sich aus Wegpunkten (dreidimensionalen Vektoren), welche in einer Liste gesammelt werden, zusammen. Mittels eines von Lightning bereitgehaltenen Objekts, dem ltVectorInterpolator, werden die Punkte in der Liste nacheinander durchlaufen und dabei linear interpoliert. Die Ausgabe der Interpolatoren für den Positions- und Orientierungspfad wird nun an die Kameraposition und -orientierung weitergegeben.

Der Interpolator benötigt für das Durchlaufen einer Strecke zwischen zwei Punkten immer dieselbe Zeit, unabhängig von der Distanz dieser Punkte. Somit ist es möglich, die Geschwindigkeit der Pfadanimation zu steuern, denn je weiter die Punkte voneinander entfernt liegen, desto schneller ist die Bewegung. Werden die Kurven des Pfades zusätzlich mit mehreren Zwischenpunkten versehen, so erscheinen sie dem Anwender bei einem Transport runder und daher auch angenehmer.

Damit die Pfade den zuvor genannten Ansprüchen genügen, wurden sie bei einem Rundgang in der virtuellen Stiftskirche einmalig zwischen den einzelnen Modellen aufgezeichnet. Die einzelnen Punkte der Position und Orientierung wurden auf dem Weg zwischen den informationsbehafteten Modellen im Sekundentakt aufgezeichnet und in eine externe Datei geschrieben. Anschließend wurden für jeden Pfad zwischen zwei Objekten die Vektoren in Listen, welche bei Programmstart geladen werden, abgelegt. Die geführte Tour beginnt am Haupteingang der Stiftskirche und führt den Anwender zuerst zum Epitaph des Johannes Brenz. Hierauf wird er zur Grabplatte Vergenhans weitergeführt und gelangt von dort an die Grafenstandbilder. Nun bringt ihn die Tour weiter in die Gruft zum Sarg. Danach führt der Weg über die Wendeltreppe hinauf zur neuen Orgel und von dort aus schließlich wieder zurück zum Eingang. Hervorzuheben ist hierbei, dass der Betrachter sich an jedem Modell zwischen den Pfaden ausführlich informieren kann. Erst wenn er auf dem Info-Menue die Schaltfläche der geführten Tour wieder aktiviert, wird der Weg fortgesetzt.

Aktiviert der Anwender bei seinem Rundgang die Schaltfläche der geführten Tour, werden die Routen des Walk- oder Fly-Objekts zum Kamera-Objekt durch neue Routen zwischen Interpolatoren und Kamera ersetzt. Befindet sich der Anwender nun beispielsweise am Brenz Epitaph (Wandtafel), werden sein aktueller Standpunkt und seine Blickrichtung als erste Werte in die jeweilige Liste der Pfade geschrieben. Anschließend wird mit der Methode generateKeyList eine Liste mit Werten errechnet, die dem Interpolator-Objekt die Reihenfolge der zu interpolierenden Punkte vorgibt. Die Listen der Pfade für Position und Orientierung werden nun an die Interpolatoren übergeben. Danach wird der Interpolator gestartet und das Kamera-Objekt und somit auch der Anwender bewegen sich automatisch zur Grabplatte Vergenhans. Diese Funktionalität erfolgt auf dieselbe Art, wenn der Anwender sich an einem der anderen informationsbehafteten Objekte befindet. Somit konnten die in Abbildung 55 mit „…“ gekennzeichneten Bereiche hier ausgelassen werden, da sich deren Inhalt analog zu den Feldern von 'Wandtafel' und 'Grabplatte' ergibt.

Um die beschriebene Funktionalität ausführen zu können, benötigt das Objekt Pathfinder die folgend aufgelisteten Felder.

```
{positionIn vec3 In}       # Aktuelle Position des Viewpoint
{orientationIn vec3 In}    # Aktuelle Orientierung des Viewpoint
{modeIn int In}            # Aktuell ausgewählter Modus
{actualAnnotationIn string In}   # Name des nächstliegenden Modells
{nextPOIIn int In}         # Schaltfläche geführte Tour aktiviert
```

clPathfinder

false	Mode == Inform && nextPOIIn == 1 : true					
	Routen, von Fly- oder Walknavigation, zur Kamera entfernen					
	Neue Routen, für Interpolatoren von Position und Orientierung zur Kamera, aufbauen					
	actualAnnotation ==					
	Wanddtafel	Grabplatte	Standbild	Gruft	Orgel	Eingang
	PathPos = ActualPos + PfadPos von Wandtafel nach Grabplatte	PathPos = ActualPos + PfadPos von Wandtafel nach Grabplatte	…	…	…	…
	PathOri = ActualOri + PfadOri von Wandtafel nach Grabplatte	PathOri = ActualOri + PfadOri von Grabplatte nach Standbild	…	…	…	…
	generate KeyList Position	generate KeyList Position	…	…	…	…
	generate KeyList Orientation	generate KeyList Orientation	…	…	…	…
	Starten der Interpolation des Bewegungspfades PathPos					
	Starten der Interpolation des Orientierungspfades PathOri					

Abbildung 55: Funktionsweise von Pathfinder.

7.5 Licht in der Stiftskirche

Die Szene der virtuellen Stiftskirche besitzt bisher vier Lichtquellen, die Umgebungslichter. Bei diesen handelt es sich um gerichtete OpenGL Lichtquellen (von Lightning bereitgehaltenes Objekt ltlightinfinite), deren Ausrichtung über alle Achsenwinkel angegeben werden kann. Die Position der Lichter im Raum spielt keine Rolle, da sich das Licht flächig und ungedämpft in der gesamten Szene ausbreitet. Jedes Licht bekommt für den jeweiligen Farbkanal einen kleinen ambienten Anteil von 0,1 und eine Helligkeit von 0,6 zugewiesen. Mit diesen Lichtern wird die gesamte Szene beleuchtet. Folgend ist das Licht light1 mit seinen Feldern und zugewiesenen Werten stellvertretend für die vier Lichter in der Szene dargestellt.

```
ltlightinfinite light1 -positionIn "0 0 0"
                       -ambientIn "0.1 0.1 0.1"
                       -orientationIn "0 -15 0"
                       -colorIn "0.6 0.6 0.6"
                       -enabled 1
```

Zusätzlich ist eine Punktlichtquelle vorhanden, die mit der Position des Anwenders über Routen verbunden ist. Das Licht (Headlight) bewegt sich daher mit dem Anwender durch die Szene. Diese Lichtquelle erhält ebenfalls einen ambienten Wert von 0,1 sowie eine Gesamthelligkeit von 0,4 pro Farbkanal. Damit ist zunächst überall in der Szene ein angenehmer Lichteindruck gegeben, welcher aber noch um weitere Lichter erweitert werden kann.

```
ltlight lightHead  -colorIn "0.4 0.4 0.4"
                   -ambientIn "0.1 0.1 0.1"
```

7.5.1 Lichtsteuerung in der Gruft mit der Klasse clLightController

Um die Lichtverhältnisse an die einzelnen Bereiche in der Szene anpassen zu können und die Gruft somit etwas dunkler als den Rest der Stiftskirche erscheinen zu lassen, benötigt man ein weiteres Objekt, den LightController. Dieser steuert den bereits beschriebenen Effekt auf folgende Art und Weise.

Anhand von Interpolatoren kann die Helligkeit der Umgebungslichter weich herunter gedimmt werden. Für die Liste der zu interpolierenden Punkte werden dem Interpolator feste Werte von 0,6 bis 0,3 pro Farbkanal zugeteilt. Das Headlight wird ebenfalls über einen Interpolator angesteuert, jedoch dunkelt dieses bis zu einem Wert von 0,2 Einheiten herab. Bewegt sich der Anwender auf seinem Rundgang durch das Modell nun hinab in die Gruft, werden, wenn dieser eine *z*-Position unter -260cm erreicht, die Interpolatoren

gestartet. Bis der Interpolator seinen letzten Punkt erreicht hat, werden die Werte laufend in die jeweiligen Felder der Lichter geschrieben. Somit dunkelt sich die gesamte Szene langsam ab. Da die Gruft jedoch ein geschlossener Raum ist, fällt es dem Anwender gar nicht auf, dass auch die anderen Teile der Stiftskirche weniger hell erscheinen. Erst wenn er die Treppen aus der Gruft wieder nach oben steigt und dabei eine *z*-Position von über -250cm erreicht, dreht sich die Funktionalität um, und es wird langsam wieder heller in der Szene. Diese Funktion benötigt keinerlei Interaktion des Benutzers, sondern wird allein durch das Eingabefeld positionIn des LightController gesteuert, welches über eine Route mit der Position des Benutzers verbunden ist.

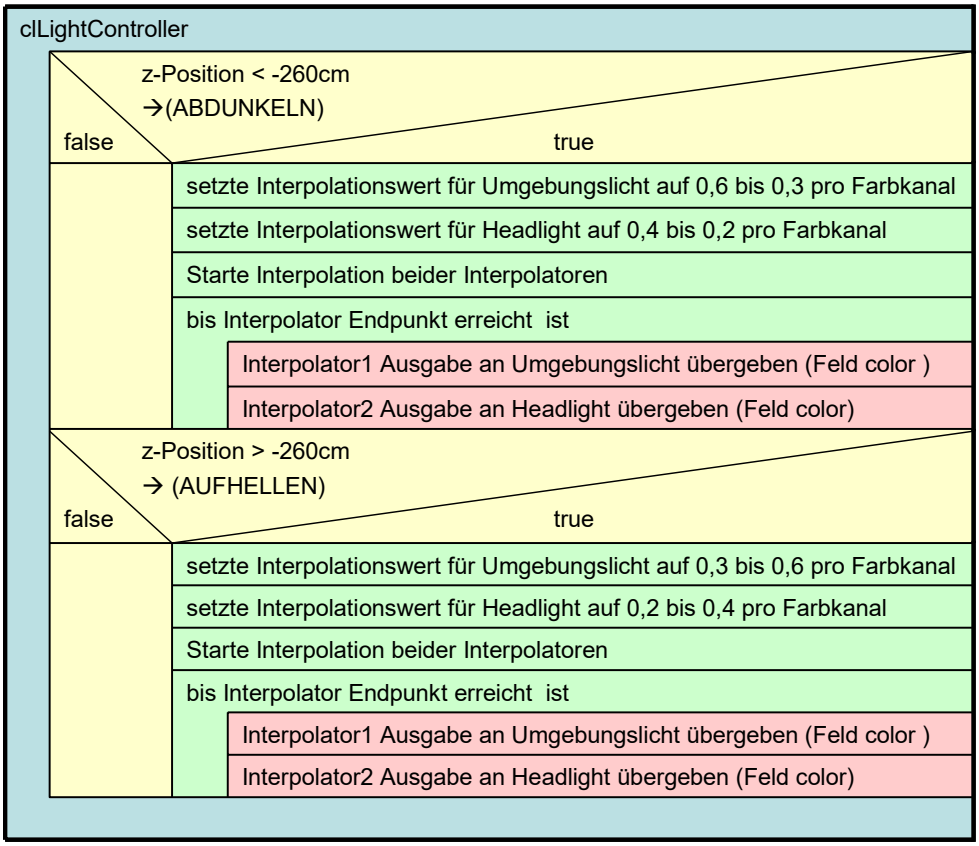


Abbildung 56: Funktionsweise des LightController.

Gesamtübersicht der Implementierung

Nachdem die Funktionen der Objekte nun ausführlich dargelegt wurden, wird im Folgenden ein Überblick gegeben, der das Zusammenspiel der elementaren Objekte untereinander aufzeigt. Damit die Übersichtlichkeit gewahrt bleibt, wurde die Grafik der Gesamtübersicht in drei Komponenten aufgeteilt. Die Linien mit Pfeilen stellen dabei die Routen zwischen den Objekten dar. Die erste Abbildung zeigt das Zusammenspiel der Objekte, die mit dem Navigationsmodus in Verbindung stehen. Die darauf folgende Abbildung zeigt die Objekte, welche den Informationsmodus definieren. In der letzten Darstellung wird der Zusammenhang der Objekte des Informationsmodus sowie der geführten Tour veranschaulicht.

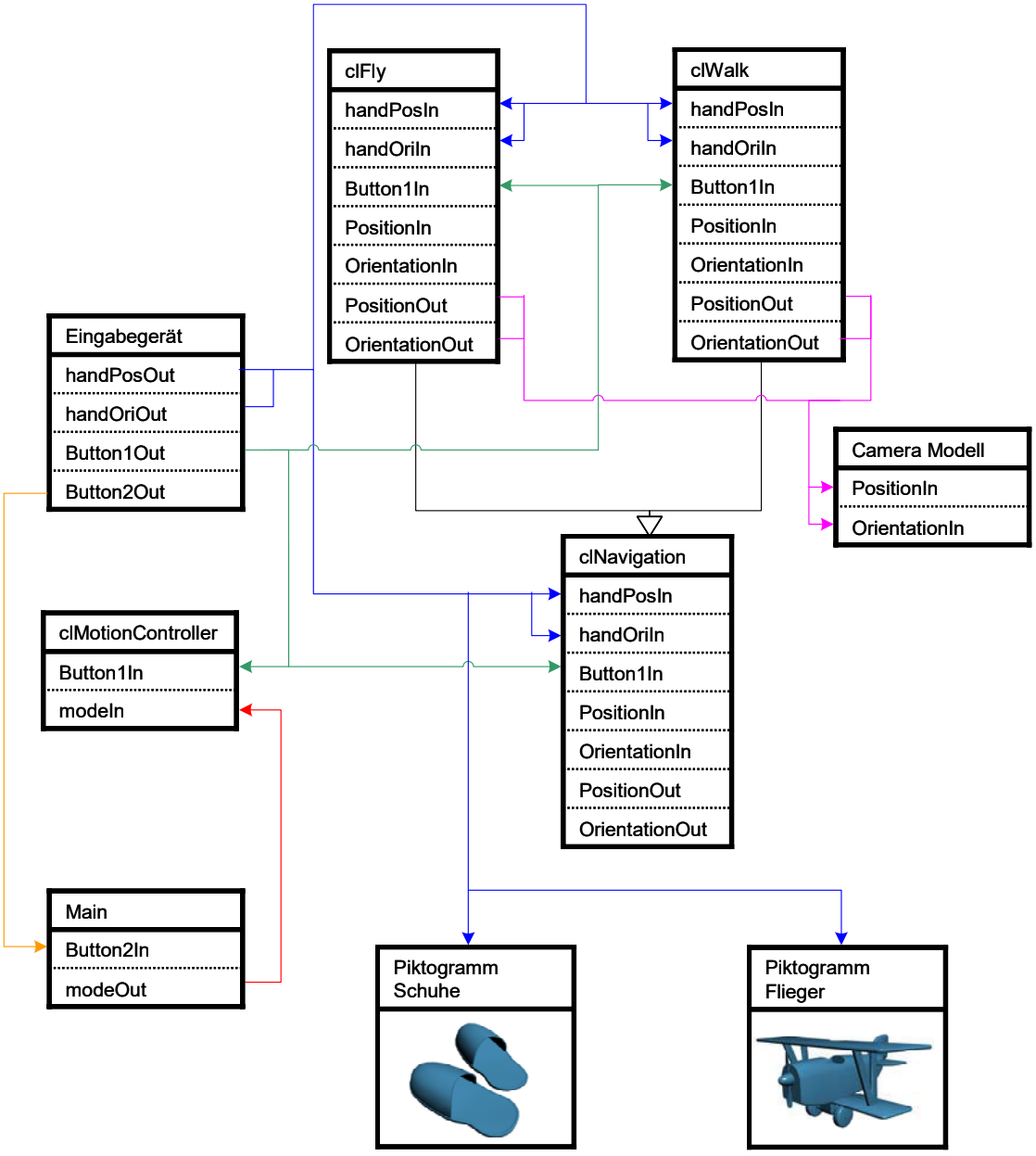


Abbildung 57: Gesamtübersicht der Navigationsobjekte.

Wie aus den Grafiken hervorgeht, erhalten die meisten Objekte Daten vom Eingabegerät und der Main-Klasse. In der Main-Klasse befindet sich der Modusumschalter, der alle Modi nacheinander durchschaltet. Diese Elemente (Main und Eingabegerät) sind in der Applikation nur einmal vorhanden. Sie wurden jedoch in alle drei Grafiken mit aufgenommen, da sie als grundlegende Steuerelemente dienen und die jeweiligen Controller und Zusatzobjekte der unterschiedlichen Modi ansteuern.

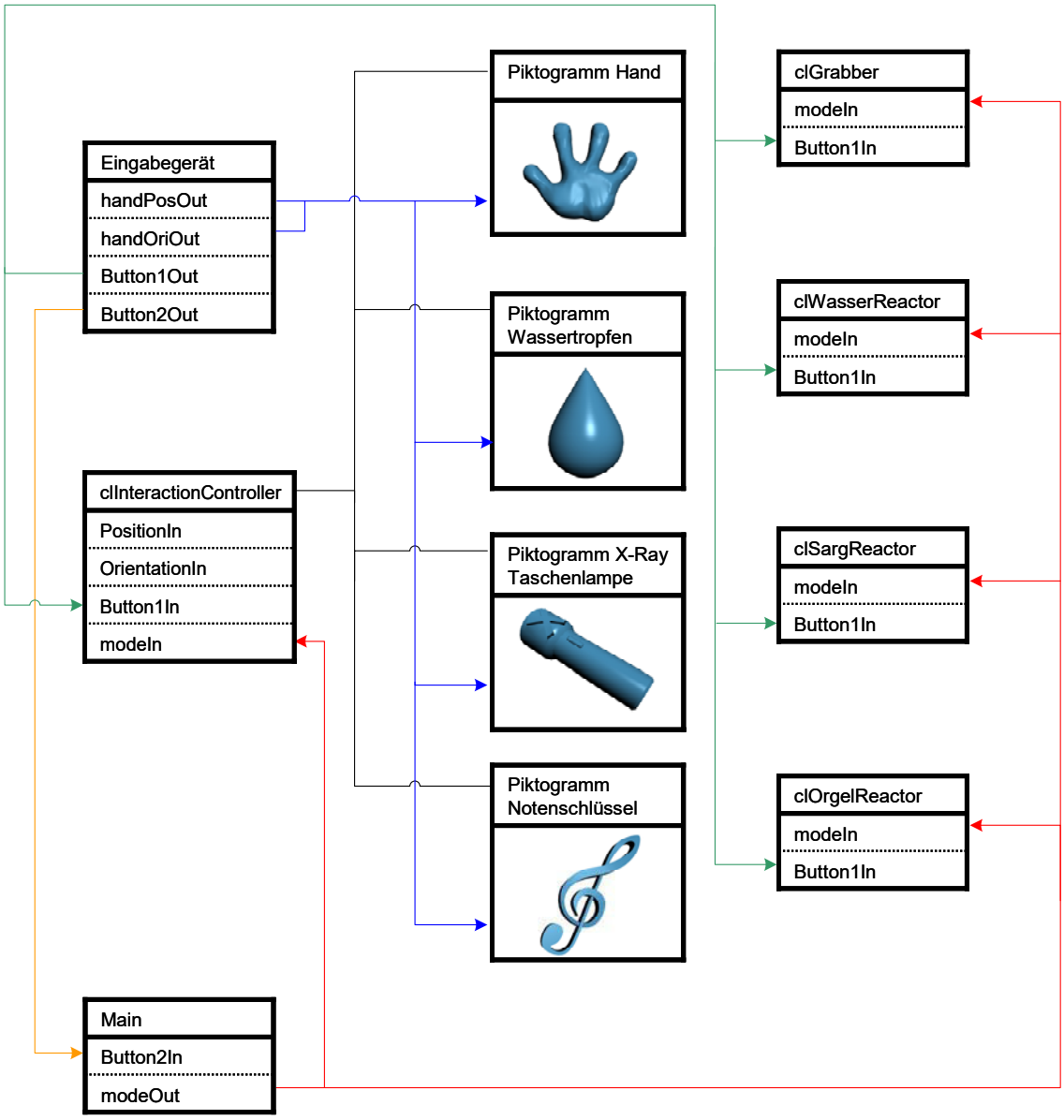


Abbildung 58: Gesamtübersicht der Interaktionsobjekte.

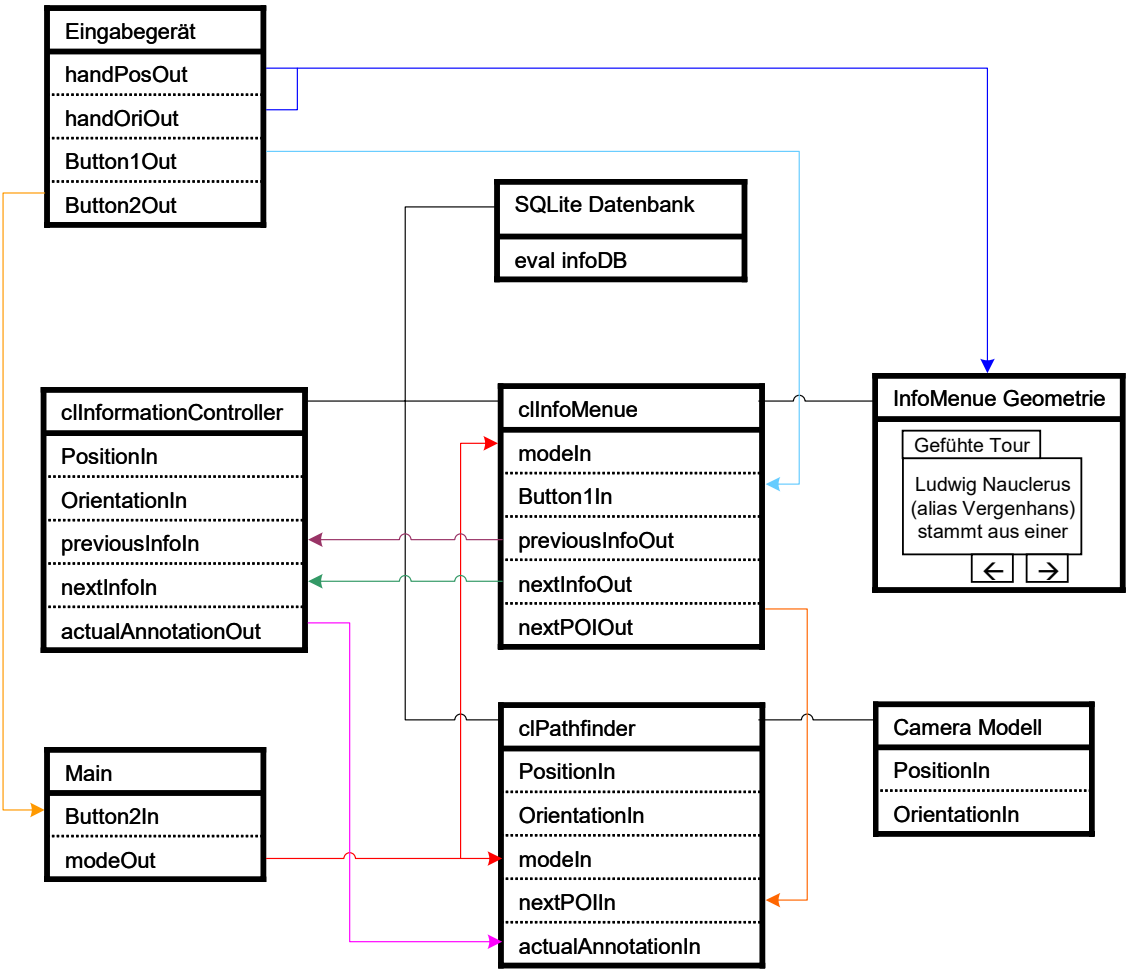


Abbildung 59: Gesamtübersicht der Informationsobjekte.

8 Ergebnisse

Die folgenden Abbildungen wurden in der CAVE™ des Fraunhofer Instituts Stuttgart aufgenommen. Sie zeigen einige eindrucksvolle Perspektiven der in dieser Arbeit entstandenen VR-Applikation.

Abbildung 60: Interaktion mit dem Buch.

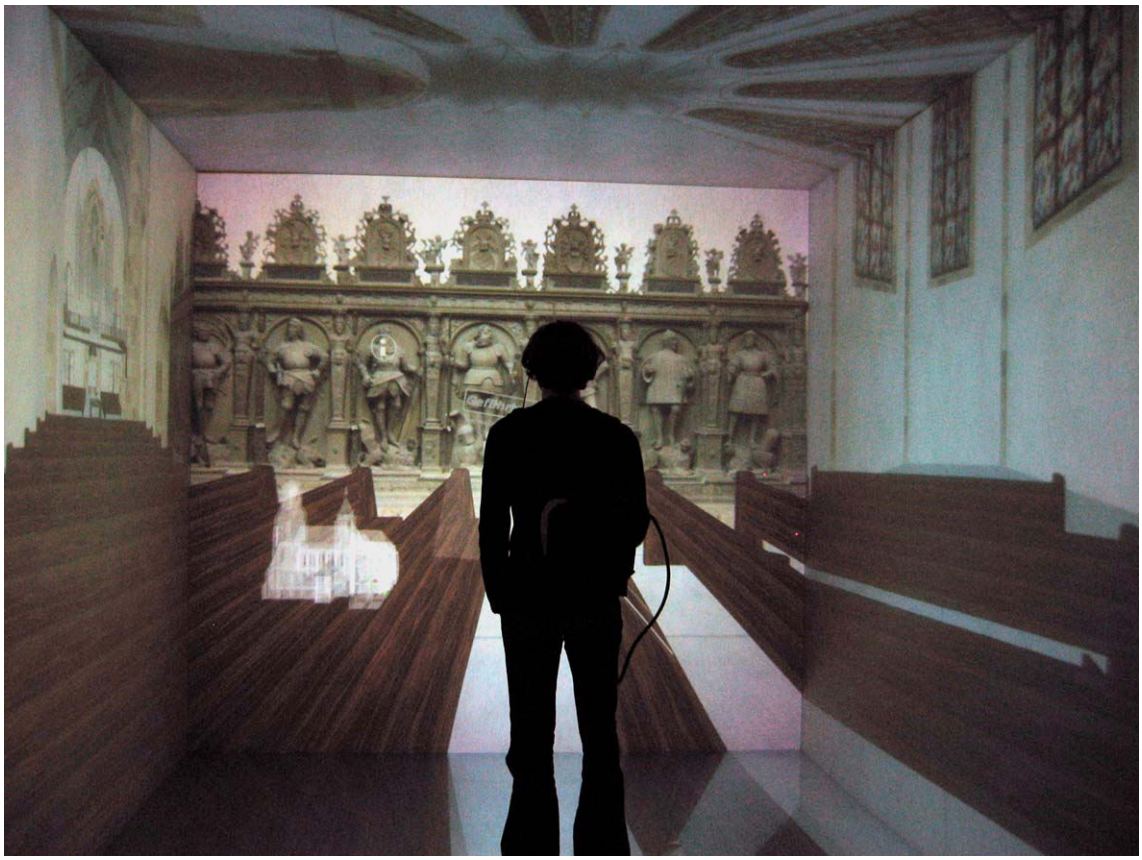


Abbildung 61: Vor dem Grafenstandbild.

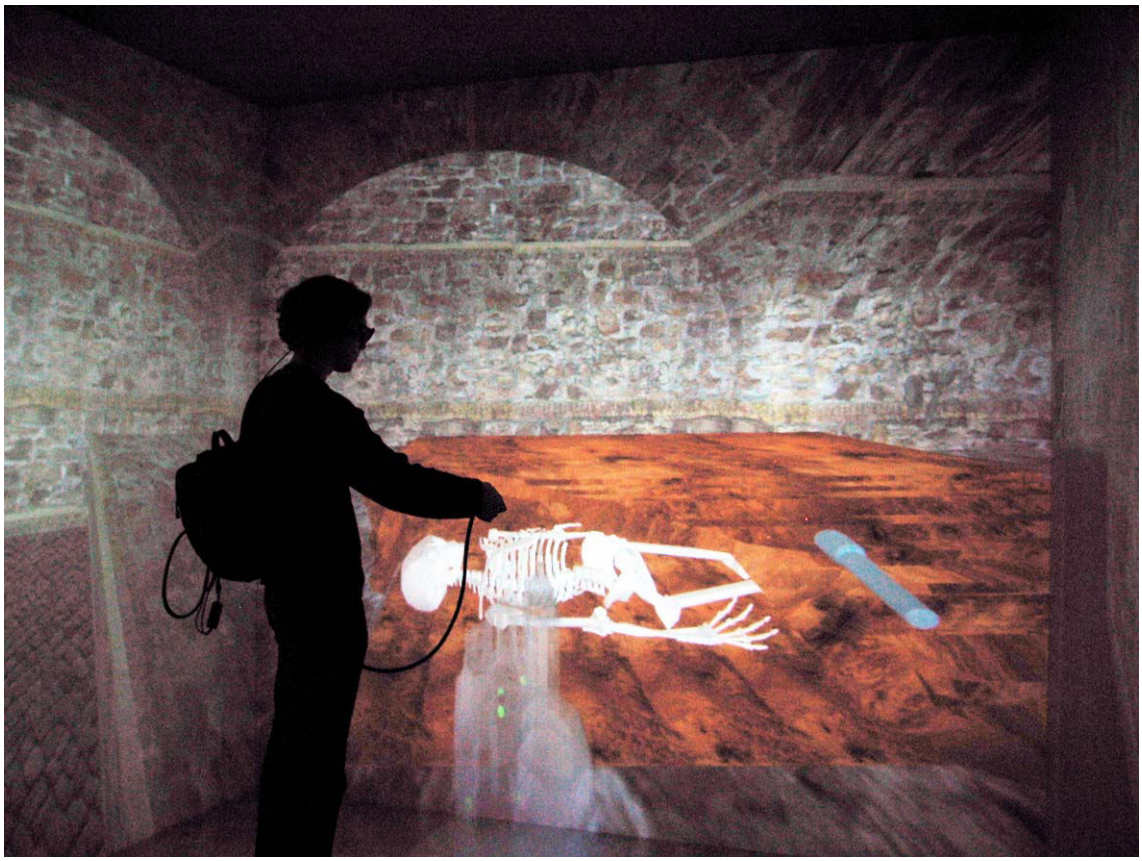


Abbildung 62: Durchleuchten der Sargwand.

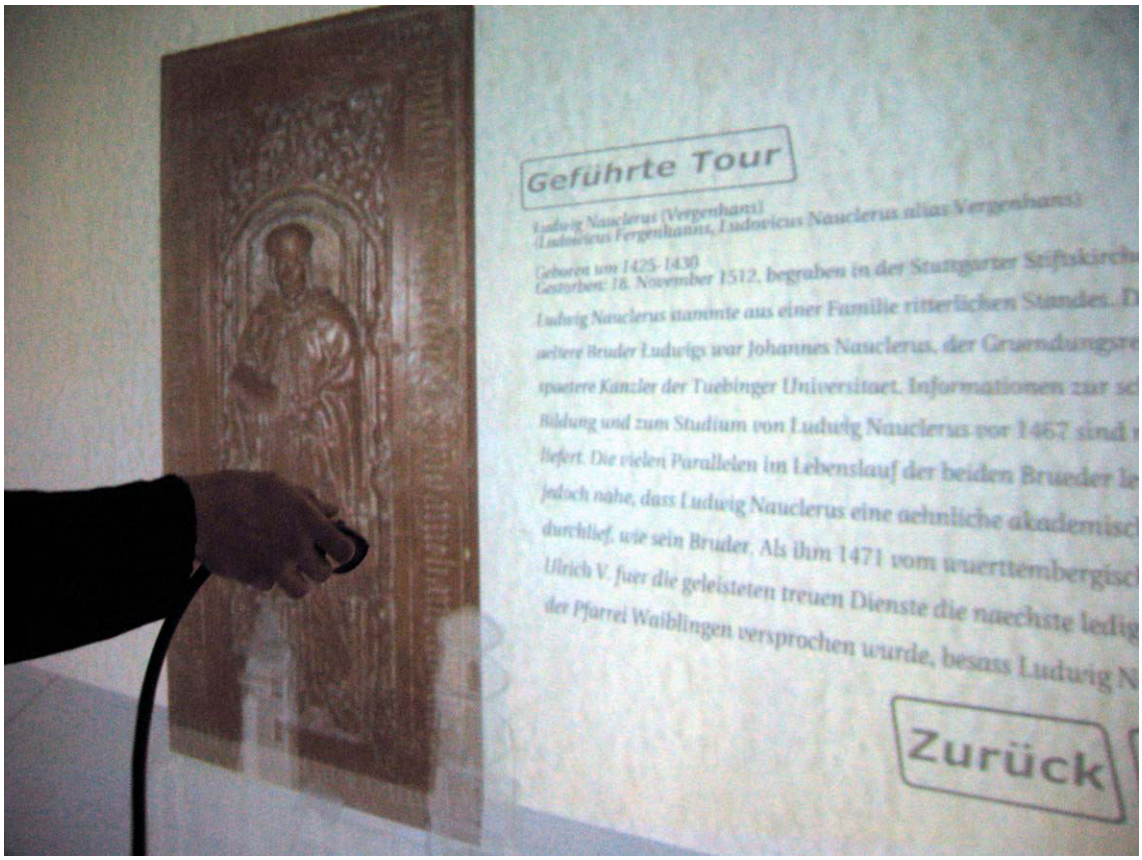


Abbildung 63: Das Informationsmenü an der Grabplatte Vergenhans.

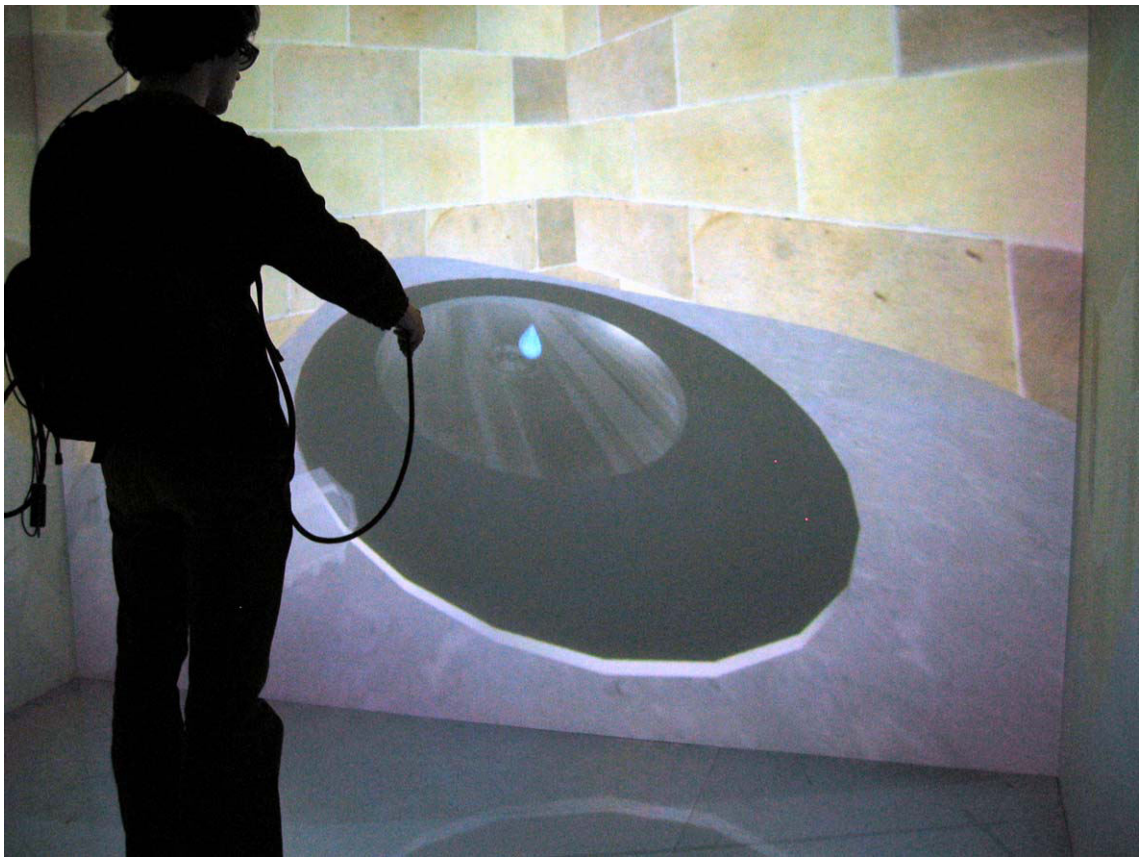


Abbildung 64: Wasseranimation am Taufbecken.

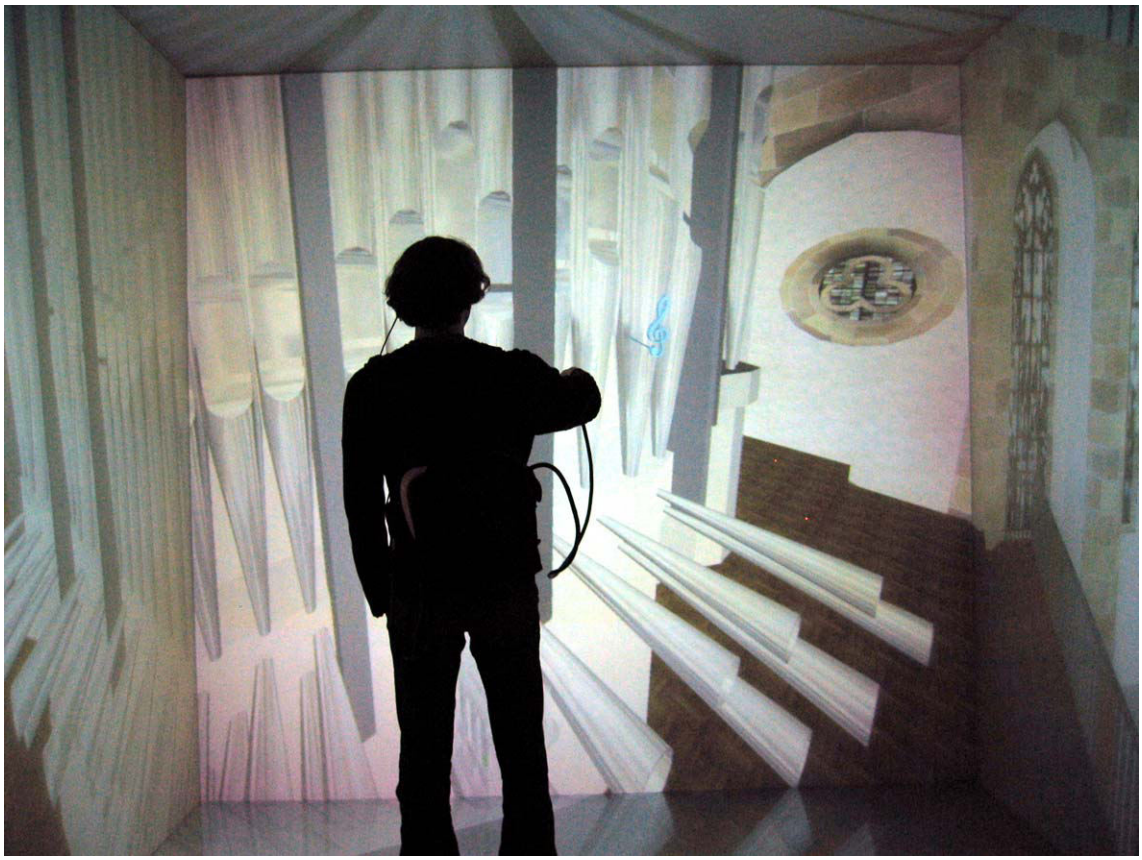


Abbildung 65: Interaktion mit der Orgel.

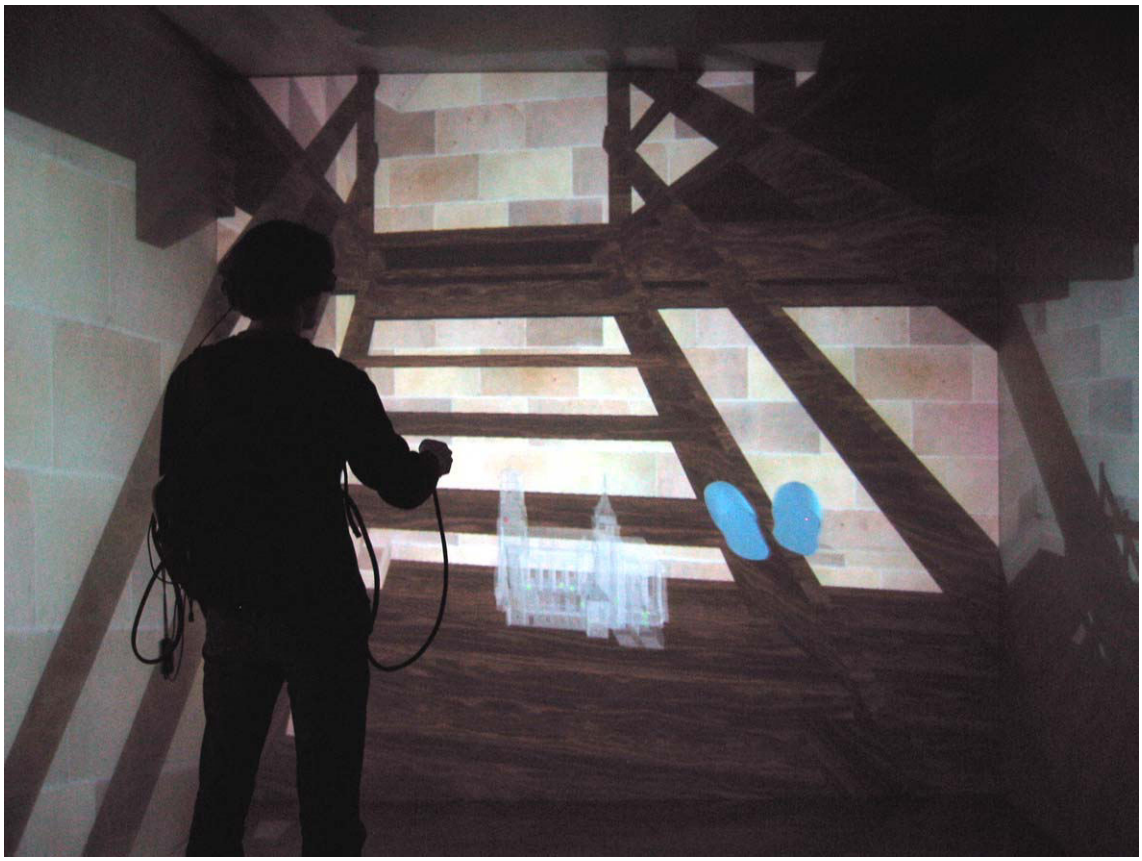


Abbildung 66: Turmaufgang und World In Miniature.

9 Ausblick

Die vorliegende Arbeit bietet eine Vielzahl von Ansätzen, den virtuellen Besuch der Stiftskirche zu erweitern. Im Folgenden sind einige ausgewählte Aspekte beschrieben.

Erweiterung der Szene mit informativen oder interaktiven Modellen:

Eine Erweiterung durch zusätzliche geometrische Modelle zur Interaktion oder Informationsvermittlung ist ohne weiteres möglich. Die hinzukommende Geometrie kann mit einem beliebigen Modellierwerkzeug erstellt und anschließend beispielsweise in VRML exportiert werden. Nun kann diese Datei zu den anderen grafischen Modellen hinzugeladen werden.

Handelt es sich bei den neuen Modellen um Geometrien, an die eine Interaktionsmöglichkeit geknüpft werden soll, kommt bei der Programmerweiterung der modulare Aufbau der Applikation zur Geltung. Dadurch kann die Anwendung leicht um Zusatzobjekte erweitert werden. Die zur Interaktion benötigte Logik wird dafür in ein neues [incrTCL]-Objekt gekapselt. Zur Ansteuerung muss die notwendige Steuerfunktion nur in die Klasse des InteractionController eingetragen werden.

Ist ein neues Modell informationsbehaftet, so wird der zugehörige Inhalt in die Datenbank eingetragen, was mit dem SQLite Browser sehr unproblematisch funktioniert. An welcher Stelle und in welchem Radius die neuen Informationen dem Betrachter bereitgehalten werden, wird im InformationController implementiert.

Die geführte Tür kann dabei auch um den zusätzlichen Pfad erweitert werden. Dafür muss ein Weg von einem existierenden informationsbehafteten Modell zum neu hinzugekommenen Modell definiert werden. Zusätzlich wird ein Pfad benötigt, der von diesem zu einem nächsten informationsbehafteten Modell führt. Die Pfade können vorab aufgezeichnet und als Variablen der Position und Orientierung abgespeichert werden. Anschließend ist es notwendig, die Klasse InformationController in ihrer Logik um ein weiteres informationsbehaftetes Modell zu erweitern.

Erweiterung der geführten Tour über die World In Miniature:

Die geführte Tour wurde bisher nur in einer relativ einfachen Logik implementiert. Eine Erweiterungsmöglichkeit lässt sich in Verbindung mit der World In Miniature (WIM) realisieren [sto95]. Ein ltSelectray kann auf die grünen Punkte der WIM, welche die Schlüsselpunkte in der Stiftskirche darstellen, sensibilisiert werden. Wählt der Anwender mit dem

Selectray einen der grünen Punkte aus, berechnet ein Wegfindungsalgorithmus den Pfad vom aktuellen Standpunkt zum ausgewählten Modell. Diese erweiterte Pfadberechnung stellt eine große Herausforderung dar, da die Lösung des Wegfindungsalgorithmus sehr komplex ist. Der Anwender kann sich an einem beliebigen Punkt in der Szene befinden und der Algorithmus muss somit von dieser Stelle aus einen realistisch begehbaren Weg zum gewünschten Objekt errechnen (siehe dazu [del01]).

Erweiterung der visuellen Detaillierung des Modells:

In der Computergrafik gibt es verschiedene Ansätze der Übertragung von Details aus der Realität auf die virtuelle Szene. Die Geometrie der Stiftskirche würde in ihrem Erscheinungsbild noch wesentlich realistischer wirken, wenn in der Szene mehr Atmosphäre durch Licht und Schatten vorhanden wäre. Lichtberechnungsverfahren wie Radiosity machen es möglich, dem Betrachter die Licht- und Schattenverhältnisse der Szene vor Augen zu führen. Eine Überarbeitung der existierenden Geometrie der Stiftskirche mit diesem Verfahren würde die Immersion des Anwenders noch um ein Vielfaches fördern.

Des Weiteren können an einigen Stellen besondere Materialien, welche spezielle Shader verwenden, eingesetzt werden. Zum Beispiel kann der Steineffekt am Altar oder das Relief der Grabplatte des Vergenhans mit der Verwendung eines Bump-Mapping Materials um ein Vielfaches detaillierter dargestellt werden.

Informationsvermittlung über Sprache:

Auch bei der Vermittlung der Informationen lassen sich neue Ansätze finden. Mit der professionellen Vertonung der Inhalte durch einen Sprecher ließe sich deren Darstellung durch Texte komplett vermeiden. Der Anwender könnte somit ebenfalls an jedem Objekt entscheiden, ob er die zugehörige Information anhören möchte. Das in der virtuellen Umgebung erschwerte Lesen entfiele dabei und der Anwender hätte mehr Kapazitäten zur Betrachtung der Szene übrig.

Geschichte erlebbar machen:

Virtual Reality ist besonders gut geeignet, um verschiedene Varianten einer Szene darzustellen. Dies ließe sich hervorragend dazu verwenden, dem Betrachter die aufregende Geschichte des Bauwerkes begreifbar zu machen. Durch den Einsatz von Modellen, die das Gebäude so zeigen, wie es vor dem Bombenhagel im Jahr 1944 ausgesehen hat sowie einem Modell im zerstörten Zustand könnte die Geschichte der Stiftskirche dem Betrachter näher gebracht werden, als dies durch geschichtliche Texte in Büchern möglich ist.

10 Anhang

10.1 Abkürzungsverzeichnis

CAVE	Cave Automated Virtual Environment
CCVE	Competence Center Virtual Environments
CPU	Central processing unit
DOF	degrees of freedom
FPS	Frames per second
GPU	Graphic processing unit
HMD	Head Mounted Display
Hz	Hertz
IAO	Institut Arbeitswirtschaft und Organisation
[incrTCL]	incremental TCL
NURBS	Non Uniform Rational B-Spline
Open SG	Open Source Scene Graph
ori	Orientation
PFA	Performer ASCII
PFB	Performer binary
POI	Point of Interest
pos	Position
SGI	Silicon Graphics, Inc
VE	Virtual Environment
VR	Virtual Reality
VRML	Virtual Reality Modeling Language
WIM	World In Miniature

10.2 Literaturverzeichnis

[aug01] O. Auge: Kleine Geschichte der Stuttgarter Stiftskirche. Leinfelden-Echterdingen: DRW-Verlag Weinbrenner GmbH & Co, 2001.

[bla98] R. Blach, J. Landauer, A. Rösch, A. Simon: A Flexible Prototyping Tool for 3D-Real-Time User-Interaction. Proceedings of the Eurographics Workshop on Virtual Environments 98, S. 195-203, 1998.

[bow01] D. A. Bowman, E. Kruijff, J. LaViola Joseph, I. Poupyrev: An Introduction to 3-D User Interface Design. Massachusetts Institute of Technology. Presence, Vol.10, No. 1, S. 96-108, 2001.

[bow04] D. A. Bowman, et al: User Interfaces. Theory and practice. Boston: Addison-Wesley, 2000.

[bru95] R. Brugger: Professionelle Bildgestaltung in der 3D-Computergrafik, Grundlagen und Prinzipien für eine ausdrucksstarke Computervisualisierung. Bonn: Addison-Wesley, 1995.

[bue03] M. Bues: Lightning: Ein System für immersive Echtzeit-Umgebungen. Skript zur Vorlesung an der FH Furtwangen. Stuttgart: Fraunhofer IAO, Competence Center Virtual Environments, 2003.

[car97] R. Carey, G. Bell, C. Marrin: ISO/IEC 14772-1:1997 Virtual Reality Modeling Language (VRML97). URL, http://www.vrml.org/Specifications/VRML97, 1997.

[cru93] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti: Surround-screen Projection-based Virtual Reality: The Design and Implementation of the CAVE. Proceedings of SIGGRAPH '93, S. 135-142, 1993.

[dec03] F. Decker, C. Runde: Fluchtpunkt, VRML-Plug-ins auf dem Prüfstand. Heise Verlag, iX 10/2003, S. 68-71, 2003.

[del00] M. A. DeLoura: Game Programming Gems. Hingham, Massachusetts: Charles River Media, 2000.

[del01] M. A. DeLoura: Game Programming Gems 2. Hingham, Massachusetts: Charles River Media, 2001.

[hec86] P. S. Heckbert: Survey of Texture Mapping. IEEE Computer Graphics and Applications, Vol. 6, Nr. 11, S. 56-67, November 1986.

[hic95] C. Hick: Physiologie. Neckarsulm: Jungjohann Verlag, 1995.

[hip04] R. Hipp: SQLite. URL, http://www.sqllite.org, 2004.

[ket93] L. Kettner: Mathematisch-Informationstheoretische Untersuchung von 3D-Metaphern. Karlsruhe: Institut für Betriebs- und Dialogsysteme, Universität Karlsruhe, 1993.

[lei97] U. Leiner, B. Preim, S. Ressel: Entwicklung von 3D-Widgets - Überblicksvortrag. München: Siemens AG. URL, http://www.mevis.de/~bernhard/papers/maerz97.html, 1997.

[mau99] R. Auf der Maur: Stencil Buffer. URL, http://www.3dconcept.ch/artikel/stencilbuffer, August 1999.

[mcl96] M. J. McLennan: Object-Oriented Programming with [incr Tcl]. Allentown: Bell Labs Innovations for Lucent Technologies, 1996.

[mor03] C. Morley: Chris Morley's Page. URL, http://www.vermontel.net/~cmorley/, 2003.

[nvi04] NVIDIA: CG Toolkit Users Manual, A Developer's Guide to Programmable Graphics. Santa Clara: NVIDIA Corporation, URL, http://developer.nvidia.com/Cg, 2004.

[pie91] L. Piegl: On NURBS: A survey. Computer Graphics and Applications, Januar 1991.

[roh94] J. Rohlfs, J. Helman: IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D-Graphics, Proc. ACM SIGGRAPH, Orlando, 1994.

[sch] L. Schmidt: Stiftskirche Stuttgart - Wahrzeichen und älteste evangelische Kirche in Stuttgart. URL, http://www.stiftskirche.de (Geschichte).

[SGI04] SGI White Paper: OpenGL Performer™ Real-Time 3D-Rendering for High-Performance and Interactive Graphics Applications. Crittenden Lane: Silicon Graphics, Inc. URL, http://www.sgi.com/products/software/performer/whitepapers.html, 2004.

[she03] W. R. Sherman, A. B. Craig: Understanding Virtual Reality. Interface, application, and design. San Francisco: Morgan Kaufmann Publishers, 2003.

[spe96] T. Sperlich: Neues in der Datenhöhle. In: Die Zeit, Nr. 47, 15. November 1996.

[sta87] M. Staufer: Piktogramme für Computer, Kognitive Verarbeitung, Methoden zur Produktion und Evaluation. Berlin: Walter de Gruyter, 1987.

[stu96] R. Stuart: Design of Virtual Environments. New York: McGraw-Hill, 1996.

[sto95] R. Stoakley, M. J. Conway, R. Pausch: Virtual Reality on a WIM: Interactive Worlds in Miniature. The University of Verginia, 1995.

[sor] T. Sorg, H. Schmidt-Glassner: Die Stiftskirche in Stuttgart. Königstein im Taunus: Langenwiesche - Bücherei.

[til83] W. Tiller: Rational B Splines for Curve and Surface Representation. IEEE Computer Graphics and Applications, September 1983.

[vel02] I. Velsz: 3ds max4, Grundlagen und Praxis der 3D-Visualisierung und -Animation. München: Addison-Wesley, 2002.

[yio01] C. Yiorgos, C. Daniel, D. Fredo, G. Ned, K. Vladlen, T. S. Claudio: Visibility, problems, techniques and applications. Course Notes for SIGGRAPH 2001 Los Angeles, California, August 2001.